DECnet–ULTRIX

Release Notes

May 1990

This manual gives you miscellaneous information and updates not included in the DECnet–ULTRIX documentation set.

Supersession/Update Information:	This is a revised manual.
Operating System and Version:	ULTRIX V4.0
Software Version:	DECnet–ULTRIX V4.0

digital™

Order Number: AA–JD82E–TE

AA–JD82E–TE
May 1990

This manual was produced by Networks and Communications Publications.

Contents

Chapter 3 Unsupported Utilities

Tables

Read Me First

The DECnet–ULTRIX V4.0 product is a new release of DECnet–ULTRIX software that builds upon the functionality of previous releases. DECnet–ULTRIX V4.0 software is a Phase IV end-node implementation of Digital Network Architecture (DNA) for the ULTRIX V4.0 operating system.

You should read this manual before you install your distribution software.

This manual contains three chapters:

- Chapter 1 describes the special considerations you must be aware of regarding LMF.
- Chapter 2 describes undocumented information regarding the DECnet–ULTRIX V4.0 software. Also describes changes made to the documentation set.
- Chapter 3 describes the three unsupported utilities (**update_nodes**, **tell**, and **nfi**) that are provided with DECnet–ULTRIX V4.0.

Chapter 1

License Management Facility (LMF) Support

DECnet–ULTRIX now supports the ULTRIX Operating System's License Management Facility (LMF). This chapter describes the special considerations you should be aware of regarding LMF.

1.1 Registering Your Software License with LMF

You must use LMF to register your license for DECnet–ULTRIX V4.0 software. If you do not register your software license, you will not be able to perform any remote access with DECnet–ULTRIX software.

If this is your first installation of DECnet–ULTRIX, you have been issued a License Product Authorization Key (PAK). Please register the data from the License PAK in the license database. The *Guide to Software Licensing* and the LMF man page, **lmf(8)**, both describe how to use LMF.

If this is not your first installation of DECnet–ULTRIX, or if you do not have your License PAK, an Installation PAK has been included with the DECnet–ULTRIX software. To register the Installation PAK in the license database, do the following:

```
# lmf register - < /usr/lib/dnet_shared/DECnet-ULTRIX.PAK RET
# lmf reset RET
```

You can register your software license before or after installing DECnet–ULTRIX.

1.2 Using Both the lmf register and lmf reset Commands

Please note that you must use both the **lmf register** and **lmf reset** commands when you register your license for DECnet–ULTRIX software. The **lmf register** command registers your software license, and the **lmf reset** command copies the license details from the License Database (LDB) to the kernel cache. (If you do not issue the **lmf reset** command, you must reboot your system to copy the license details to the kernel cache.)

For more information, refer to the *Guide to Software Licensing*.

Chapter 2

General Software Notes

This chapter describes special considerations and undocumented software features in DECnet–ULTRIX V4.0 software. It consists of the following sections:

- Installation and configuration issues
- Programming issues
- Special considerations for remote access
- Changes to the DECnet–ULTRIX documentation
- Associated documentation

2.1 Installation and Configuration Issues

This section describes some details that you should be aware of before installing the DECnet–ULTRIX V4.0 software.

2.1.1 Installing DECnet–ULTRIX Software onto VAX and RISC Systems

You can install DECnet–ULTRIX V4.0 software onto any VAX or RISC system supported by the ULTRIX V4.0 system. The only difference between the two types of installations is the software subset names:

System Type	DECnet–ULTRIX Subset Names Prefix	ULTRIX Subset Names Prefix
VAX	DNU	ULT
RISC	DNP	UDT

2.1.2 DECnet-ULTRIX Software Included on ULTRIX V4.0 RA60 and CDROM

For customers who require RA60 and CDROM distribution media, the DECnet-ULTRIX software is now included on the ULTRIX V4.0 RA60 and CDROM distribution.

To mount your RA60, follow the instructions in the *DECnet-ULTRIX Installation* manual for mounting a CDROM.

2.1.3 DECnet Programming Examples

The DECnet programming examples in the *DECnet–ULTRIX Programming* manual also appear on-line in **/usr/examples/decnet**.

2.1.4 DECnet Subroutine Locations

The DECnet subroutines are located in their own library (**libdnet.a**). When you build programs that use these routines, you must specify **-ldnet** in the command line or makefile. Versions of the libraries suitable for use by **lint** are contained in the unsupported subset (DNUUNS400 for VAX systems or DNPUNS400 for RISC systems).

2.1.5 Object mail11

The **mail11** and **mail11d** programs were renamed to **mail11v3** and **mail11dv3**, respectively, in DECnet–ULTRIX V3.0. These new programs support V3.1 of the DECnet MAIL-11 protocol. This support includes CC (Carbon Copy) lines, Block Mode transfer, and DDIF document exchange with other DECnet implementations that also support it. The **sendmail** mailer definition and object database definitions changed.

If you are planning to use your DECnet–ULTRIX V2.* object database, you must modify the definition of the **mail11** object as follows:

```
Number          = 27
File            = /usr/etc/mail11dv3
Default User    = daemon
Type            = Sequenced Packet
Accept          = Deferred
```

Refer to the *DECnet–ULTRIX NCP Command Reference* for the appropriate command syntax.

To use your ULTRIX V2.* **sendmail** configuration file, you must update it to include the new definition and mailer-specific rewrite rules for **mail11**, which you can find in the new configuration file.

You can use the new **mail11v3** mailer only with the supported **sendmail** that comes with ULTRIX V4.0 due to modifications that were necessary for the mailer to be compatible with the DECnet MAIL-11 protocol.

2.1.6 load and trigger Commands

Ensure that the ULTRIX MOP subset (ULTMOP400 for VAX systems or UDTMOP400 for RISC systems) has been installed onto your system. If it has not, install it before you use the **ncp load** and **ncp trigger** commands.

2.1.7 Installing the ULTRIX MOP Subset

WARNING

Do not install the ULTRIX MOP subset (ULTMOP400 for VAX systems or UDTMOP400 for RISC systems) while DECnet is running, as this will result in loss of the volatile nodes data base. If you need to install

MOP after installing DECnet, you should turn DECnet off, install the subset, and turn DECnet back on. If the volatile nodes database has already been lost, it can be recovered by entering the command, **ncp set known nodes all**.

2.1.8 Layered Products that Require DECnet–ULTRIX Software

If you are installing a layered product on DECnet–ULTRIX V4.0 that requires a previous version of DECnet–ULTRIX, you must manually create a lock file to allow the installation to succeed.

Two layered products that require lock files are

- DECnet/SNA ULTRIX 3270 Terminal Emulator
- DECwindows DECnet/SNA 3270 Terminal Emulator for ULTRIX

To determine if your layered product requires DECnet–ULTRIX, refer to its product requirements.

To create the lock file for a VAX system, log in as superuser and enter this command:

```
# cp /dev/null /usr/etc/subsets/DNUBASE030.lk
```

To create the lock file for a RISC system, log in as superuser and enter this command:

```
# cp /dev/null /usr/etc/subsets/DNPBASE030.lk
```

2.1.9 syslog

The **dnet_spawner** can log information about various DECnet activities. For example, DECnet startups and shutdowns are recorded at level LOG_INFO. If this information is missing from the appropriate log files, the omission is probably due to a race condition between the spawner and **syslog**.

Whether or not this problem is seen on a given system depends on the amount of time between DECnet startup and the starting of syslog in **rc.local**. If the information is important to the system administrator, there are several possible solutions:

- Move the DECnet start-up code closer to the "local daemons" section in **rc.local**. (You can use **lcp** commands after DECnet startup.)
- Turn DECnet off and then back on at the end of **rc.local**.
- Start **syslog** immediately after the **ifconfig** lines, rather than in the "local daemons" section.

2.1.10 Defining the fal Object

The file access listener (**fal**) supports parameters that you can change by setting environment variables before executing the program **/usr/etc/fal**.

To change **fal** parameters, define the **fal** object so that it points to an executable shell script that sets the environment variables to the desired values and then executes **/usr/etc/fal**. If you specify a relative file name in the object database entry for **fal**, the file name will be relative to the user directory under which **fal** is running.

This command causes the DECnet object spawner to run the file **/etc/fal_params** when **fal** receives a connection request:

```
% ncp set object fal file /etc/fal_params RET
```

This command causes the DECnet object spawner to search for the file **fal_params** in the user's home directory when **fal** receives a connection request:

```
% ncp set object fal file fal_params RET
```

If the spawner does not find **fal_params** in the user's home directory, it looks for the file in **/etc/fal_params**.

The following example assigns default values to the environmental variables used by **fal**. Note that the last line in the file executes the standard file, **/usr/etc/fal**, which does the actual work.

```
#! /bin/csh -f
# explicitly set fal parameters to their default values
#
setenv fal trace 0
setenv fal tracefile FAL TRACE
setenv fal clobber 0
setenv fal unmask 91
setenv fal print "/usr/ucb/lpr"
exec /usr/etc/fal
```

Table 2–1 lists the **fal** parameters and their values:

Table 2–1: fal Parameters and Their Defaults

Parameter	Value	Default	Effect
fal_trace	0 or 1	0	Turns on tracing of DAP messages to **fal_tracefile** (if set to 1).
fal_tracefile	string	FAL_TRACE	Sets the name of the DAP message trace file (used if **fal_trace** = 1).
fal_clobber	0 or 1	0	Determines whether an attempt to create a file whose name matches an existing file will result in an error (0), or will be allowed (1).
fal_umask	decimal	91 (0133)	Sets **fal's** default protection mask for file creation (used when creating files from non-ULTRIX systems).
fal_print	string	**/usr/ucb/lpr**	Sets the name of file to be executed when a remote system requests that a file be printed.
fal_inet	nonunix	not set	May be set to nonunix for DECnet–Internet Gateway to force Internet. Connection to ASCII mode data transfer.
fal_ascii	crnul	not set	May be set to crnul for DECnet–Internet Gateway to force handling of CR NUL Internet connection. Valid only if **fal_inet** is set to nonunix.

2.1.11 Maximum Links Parameter

The **maximum links** parameter, which specifies the maximum number of active logical links for the executor node, can now be set to any number from 1 to 1024. Previously, you could have only 256 active links.

2.2 Programming Issues

This section describes special programming considerations.

2.2.1 Recompiling Object Files

You must recompile all object files compiled before DECnet-ULTRIX V4.0 if those object files use or reference any of the following structures:

- **nodeent** structure defined in **<netdnet/dnetdb.h>**
  (i.e., the **getnodeent** library routines)
- **dn_naddr** structure defined in **<netdnet/dn.h>**
- **sockaddr_dn** structure defined in **<netdnet/dn.h>**

2.3 Special Considerations for Remote Access

This section describes limitations in remote node access. These limitations include restrictions on remote connections, remote login, file transfer, and mail transfer.

2.3.1 Connections to Phase III and Phase IV DECnet Nodes

Support for outgoing connections from DECnet–ULTRIX to DECnet Phase III systems is limited to DECnet–VAX systems, and DECnet–RT systems with patches. Patches are also available for Phase IV systems to which you have difficulty establishing connections. Contact your software support representative if you have problems establishing connections with Phase III or Phase IV nodes.

2.3.2 File Transfers from VMS V4.4 and V4.5 Systems

If you are logged on to a VMS V4.4 or V4.5 system and you have copied a file from an ULTRIX node, before you copy it back to an ULTRIX system you must use the Convert utility in VMS to convert the file to stream-lf format. The VMS V4.5 release notes describe this procedure in more detail.

If you use the **dcp** command to copy a file from an ULTRIX system to a VMS V4.4 or V4.5 system, you do not have to use the Convert utility before copying the file back to an ULTRIX node.

2.4 Changes to the DECnet–ULTRIX Documentation Set

The following sections summarize the changes made to certain DECnet–ULTRIX documents.

2.4.1 *DECnet–ULTRIX Use*

This manual has been updated for Version 4.0 of the DECnet–ULTRIX software product. The *DECnet–ULTRIX Use* manual was part of the *DECnet–ULTRIX User's and Programmer's Guide*. All end user information is included in this manual; all programming information is included in the new *DECnet–ULTRIX Programming* manual.

2.4.2 *DECnet–Internet Gateway Use and Management*

This manual has been updated for release with Version 4.0 of the DECnet–ULTRIX software product. The changes include:

- Updates to technical information, including new troubleshooting information for Gateway managers.
- More tutorial and conceptual information in Chapters 2 and 3, to make the Gateway easier to use for both new and experienced users. The sections are organized by tasks, rather than by commands.

2.4.3 *DECnet–ULTRIX Programming*

This manual has been updated for Version 4.0 of the DECnet–ULTRIX software product and includes the following changes:

- The *DECnet–ULTRIX Programming* manual was formerly the *DECnet–ULTRIX User's and Programmer's Guide*. The user information has been removed from this manual and is now in the *DECnet–ULTRIX Use* manual.
- The manual pages describing the maintenance commands have been removed from this manual and added to the *DECnet–ULTRIX Network Management* manual.
- The Data Link Interface (DLI) information has also been removed from this manual and is now in the *ULTRIX Guide to the Data Link Interface* manual.
- Some of the general ULTRIX networking concepts, previously described in Chapter 3 of the *DECnet–ULTRIX User's and Programmer's Guide*, have been integrated with the 4.0 version of the *ULTRIX Network Programming Guide*.
- The numbering scheme for the reference sections in Part II of this manual corresponds to the numbering scheme for the DECnet–ULTRIX on-line manual pages. For example, system calls are described in Section 2dn, and each DECnet–ULTRIX system call name appears in the header followed by a (2dn).

2.4.4 *DECnet–ULTRIX Network Management*

This manual has been updated for Version 4.0 of the DECnet–ULTRIX software product, and includes the following changes:

- The *DECnet–ULTRIX Network Management* manual was formerly the *DECnet–ULTRIX Guide to Network Management*. The command reference information has been removed from this manual and is now in the *DECnet–ULTRIX NCP Command Reference* manual.
- Manual pages describing maintenance commands (Section 8dn) have been added to this manual.

- Descriptions of the **dts/dtr** node-level and circuit-level tests have been added.

2.4.5 *DECnet–ULTRIX NCP Command Reference*

This manual has been updated for Version 4.0 of the DECnet–ULTRIX software product, and includes the following changes:

- The *DECnet–ULTRIX NCP Command Reference* manual was formerly part of the *DECnet–ULTRIX Guide to Network Management*. This new manual contains **ncp** command descriptions as well as quick-reference information.
- Commands used for permanent and volatile databases are described on separate pages in the **ncp** command reference pages; for example, the **list node** and **show node** commands each have their own description.

2.5 Associated Documentation

For detailed information about ULTRIX system management and network management, see the following ULTRIX documentation:

- *Introduction to System and Network Management*
- *Guide to System Environment Setup*
- *Guide to Networking*
- *Guide to the Network File System*
- *Guide to the Yellow Pages Service*
- *Guide to the System Shutdown and Startup*
- *Guide to System Backup and Restore*
- *Guide to System Configuration File Maintenance*
- *Guide to System Disk Maintenance*
- *Guide to the Error Logger System*
- *Guide to System Crash Recovery*
- *Guide to System Exercisers*
- *Guide to the Diskless Management Services*
- *Guide to the uucp Utility*
- *Guide to the Remote Installation Service*
- *Guide to Ethernet Communications Servers*
- *Guide to IBM Terminal Emulation*
- *Guide to the BIND Service*

Chapter 3

Unsupported Utilities

This chapter describes utilities that Digital Equipment Corporation supplies on an "as-is" basis. These utilities are furnished for general customer use. However, support is not offered for these utilities, nor are they covered under any of Digital's support contracts.

3.1 Unsupported Utilities Subset

The unsupported utilities provided with DECnet–ULTRIX V4.0 software are contained in their own subset (DNUUNS400 for VAX systems or DNPUNS400 for RISC systems).

3.2 update_nodes

The **update_nodes** utility lets you update the local nodes database with information from a remote DECnet nodes database.

To run the **update_nodes** utility use this command format:

update_nodes *node-id*

where *node-id* is the DECnet node name or DECnet node address of the remote node whose database you are using for the update.

Non-DECnet entries, such as terminal servers, and parameters, such as down-line load parameters, are preserved except where conflicts occur. A conflict occurs when a node name received from the remote node already exists in the current database, but the node numbers do not match. In this case, the information received from the remote node is entered in the new database. However, any parameters associated with the node name are not entered. A record of all conflicts and any dropped parameters is kept in the file **/usr/lib/dnet/update_nodes.log**.

The **update_nodes** utility has a no conflicts option (**-n**). When you specify **-n**, the old database is replaced only if no conflicts are encountered. If any conflicts are discovered, the new database is left in the file **/usr/lib/dnet/nodes_p.new**. The system manager can then manually rename the new database to **nodes_p** after checking the log file **/usr/lib/dnet/update_nodes.log** and resolving the conflicts.

To run **update_nodes** with the **-n** option, use this command format:

update_nodes -n *node-id*

where *node-id* is the DECnet node name or DECnet node address of the remote node whose database you are using for the update.

The **update_nodes** utility also has a **-f** (fast) option. When run without the **-f** option, **update_nodes** keeps any previously defined parameters (such as down-line load parameters) in the nodes database. With the **-f** option, the utility runs significantly faster, but the nodes database contains only node names and numbers; all the other parameters, if any, are lost.

To run **update_nodes** with the fast option, use this command format:

update_nodes -f *node-id*

where *node-id* is the DECnet node name or DECnet node address of the remote node whose database you are using for the update.

WARNING

Always use the **-n** option if you are using your system as one of the following:

- A server for diskless clients
- A **rls** server
- A load host for DECserver terminal servers, DECrouter routers, or Internet Portal products

If you do not use the **-n** option, important parameters may be dropped from the database if any conflicts occur when the database is updated. If this happens, you can find a copy of the old database in the file **/usr/lib/dnet/nodes_p.sav**. You can copy this file to **/usr/lib/dnet/nodes_p** to restore the parameters. If you have not restarted DECnet since you last ran **update_nodes**, you can copy the file **/usr/lib/dnet/nodes_v** to **nodes_p**.

3.3 tell

The **tell** utility lets you execute commands on a remote DECnet–ULTRIX or DECnet–VAX node. If the remote system does not have **tell**, copy the file **/usr/examples/decnet/TELL.COM** to the directory associated with the username/password to which you are connecting and name the file **TELL.COM**. (The username and password give you the access and privileges you need to execute the remote command.) You can use the **tell** command in this format:

tell *node-id*[*/username/password*] *command*

where

node-id	is the DECnet node name or DECnet node address of the remote node at which you want to execute the command.
username	is the name for a remote user account.
password	is the password for a remote user account.
command	is the remote command that executes on the remote node.

In this example, the SHOW TIME command executes on the VMS node TUPELO:

```
% tell tupelo sho time RET
10-MAR-1989 10:25:15
```

To execute more than one command on a remote system, simply type **tell** and press RET. The **tell** utility prompts you for a node name or address and then for the commands that you want to execute. When you finish entering commands, press CTRL/D to return to the local shell. For example:

```
% tell RET
node? tupelo RET
TUPELO > sho time RET
10-MAR-1989 10:25:25
TUPELO > sho users RET
...
TUPELO > CTRL/D
%
```

You can enter a local command during a **tell** session by beginning the command with a tilde (~) or by typing a tilde (~) and RET, which puts you in local mode. You receive this prompt when you enter local mode:

```
~LOCAL >
```

where LOCAL is the name of the local node. For a list of local commands, type a question mark (?).

NOTE

With **tell** you can use only single line commands that generate output. The **tell** utility does not let you pass control to another program, such as **mail**. If you use **tell** to run a program and the program issues a read request, the request fails. Also, you cannot use **tell** to run cursor-manipulating programs.

3.4 nfi

The **nfi** utility tests network programs. This utility accepts commands that let you do the following:

- Connect to a program on a remote system
- Bind a name to a socket and wait for a connection
- Build and send data and interrupt messages
- Receive and display data and interrupt messages
- Close the connection

You can use **nfi** to debug pairs of programs that use a protocol; you can debug each end of the protocol separately.

To display the **nfi** commands, enter **nfi** and type **help**:

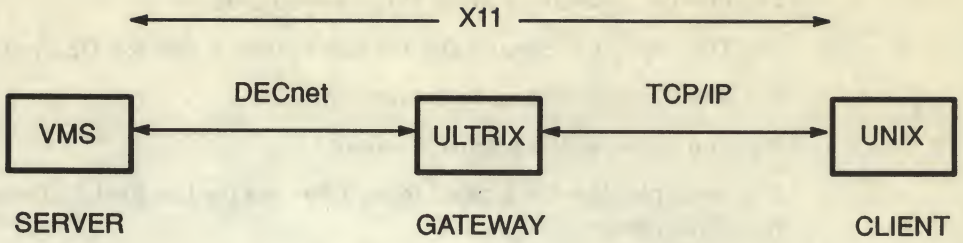
```
% nfi RET

NFI (Network Functions Interface)
  V1.01 25-Apr-85
Local node: NERTZ (4.83)

nfi> help RET
       *** nfi (network functions interface) V1.01 ***
ABORT [data {data}..]
ACCEPT [DEFER] {wait_time}
BIND "objnam" | objnum
CONNECT node[["access"][::]] OBJECT {"objnam" | objnum} ...
          [data {data}..]
CONNECT {REJECT | ACCEPT} [data {data}..]
CONVERT data
DISCONNECT [data {data}..]
EXIT
HELP | ?
RECEIVE [WAIT seconds]
SEND DATA [data..]
SEND INTERRUPT data [data..] (16 bytes maximum)
SET DISPLAY DECIMAL | OCTAL | HEXADECIMAL  | BINARY ...
          | ASCII | RAD50
SET RECEIVE ON | OFF
SET SOCKET socknum | OPTION | DOMAIN | TYPE | PROTOCOL | ...
          | BLOCKING | NONBLOCKING
SHOW LAST SEND | RECEIVE
SHOW RECEIVE | SOCKET | DISPLAY
node: area.node | node_name
@'filename' - command file
data: decimal | .decimal | \octal | #binary | %rad50 ...
          | $hex | "ascii string"

nfi> exit RET

 Exiting...
%
```

Note that **nfi** polls for both normal and out-of-band messages when you use the **recv** command.

3.5 gatewayd

The **gatewayd** unsupported utility lets you use an ULTRIX system as a gateway to swap transports for an application protocol.

For example, a client on a TCP/IP-only UNIX system might want to communicate with a server on a DECnet-only VMS system. Even if client and server both use the same protocol (for example, X11), the lack of a common underlying transport prevents communication. However, if you insert an ULTRIX system with both DECnet and TCP/IP between the two, it can act as a gateway, as shown in this figure:



LKG–3783–90I

The ULTRIX system acting as gateway must run some program that swaps application data from DECnet to Internet and vice versa. The **gatewayd** utility performs such a function.

gatewayd employs 2 sockets, one in the DECnet domain and one in the Internet domain. Once the necessary connections are established, **gatewayd** simply reads whatever is available on the DECnet socket and writes it to the Internet socket. Similarly, **gatewayd** reads whatever is available on the Internet socket and writes it to the DECnet socket.

There are two types of connections, as follows:

Connection	Client System	Server System
VMS to UNIX	VMS	UNIX
UNIX to VMS	UNIX	VMS

The following sections describe how to establish each type of connection.

3.5.1 Establishing the VMS-to-UNIX Connection

In this case (a VMS-to-UNIX connection), the VMS client establishes a connection to a specially defined object on the gateway ULTRIX system which invokes **gatewayd** and arranges for it to connect to the desired server on the UNIX system.

The following example details the steps involved for an X-protocol gateway. The procedure for other protocols would be similar. All steps are performed on the ULTRIX system that is acting as a gateway.

1. Define a new object in the DECnet object database.

   When using DECnet, X clients request connections to objects named **X$X0**, **X$X1**, **X$X2**, and so on, where each object corresponds to an X server on a given machine. A single user workstation would normally have a single server called **X$X0**. A dual-headed workstation would have 2 servers called **X$X0** and **X$X1**. When you define the new object in the DECnet database, pick a name that is not already in use (such as **X$X2**) and define it using **ncp** as shown below:

   ```
   ncp> set obj X$X2 file /usr/etc/xgate2 type stream \
   default user guest
   ```

2. Create the file specified in the **ncp** command above.

The file will be a shell script which invokes **gatewayd** with the appropriate arguments. **gatewayd** takes three arguments:

- The type of socket on the remote system (**-inet** for DECnet to TCP/IP)
- The TCP/IP system to connect to
- The name of the service desired

For example, the file **xgate2** looks like this for the first X11 server on Internet host **tcpsystem**:

```
#! /bin/sh
exec /usr/examples/decnet/gatethru/gatewayd -inet tcpsystem X0
```

3. Define the service **X0** in **/etc/services**.

   X11 servers use port numbers 6000 and up (that is, the first server listens on 6000, the second on 6001, and so on). X10 servers use ports 5800 and up. In order to connect, for example, to the first X11 server, add the following line to **/etc/services**:

```
X0 6000/tcp # X11 server port # for 1st display
```

The VMS-to-UNIX connection is now completed. The VMS client is invoked to use the **X$X2** display on the gateway; what the client actually uses is the X0 display on the UNIX system **tcpsystem**.

3.5.2 Establishing the UNIX-to-VMS Connection

You can also connect a client on a TCP/IP-only system to a server on a DECnet-only system. Once again using X11 as an example, the steps are as follows:

1. Edit the **/etc/services** file.

   Pick an X11 port number that is not in use and some identifier to associate with the VMS node where the desired X server resides. For example, to connect to the first X11 server on VMS node VMSNOD, add the following line to **/etc/services**:

```
vmsnod-X11-display0 6002/tcp  # X11 server, 1st display
```

2. Edit the file **/etc/inetd.conf**.

   You must establish a connection between the identifier **vmsnod-X11-display0** in the **/etc/services** file and an invocation of **gatewayd.c** with appropriate arguments.

   gatewayd takes three arguments:

   - The type of socket on the remote system (**-dnet** for TCP/IP to DECnet)
   - The node to connect to
   - The name of the service desired

   This is done with an entry in **inetd.conf**:

```
vmsnod-X11-display0 stream tcp nowait \
/usr/examples/decnet/gatethru/gatewayd gatex -dnet vmsnod X$X0
```

3. Reinitialize the **inet** daemon.

   You can send the daemon a HANGUP signal, kill and restart it, or reboot.

The UNIX-to-VMS connection is now complete. The UNIX client is invoked to use the second display on the gateway; what it actually uses is the first display on VMS node VMSNOD.

NOTE

gatewayd is contained in the DECnet–ULTRIX Unsupported Software subset. On your system, **gatewayd** is in the **/usr/examples/decnet/gatethru** directory, which contains four files:

gatewayd	Executable file for the **gatewayd** unsupported utility
gatewayd.c	Source code for the **gatewayd** unsupported utility
Makefile	The makefile for **gatewayd.c**
README	Description of **gatewayd**

HOW TO ORDER ADDITIONAL DOCUMENTATION

DIRECT TELEPHONE ORDERS

In Continental USA
call 800–DIGITAL

In Canada
call 800–267–6215

In New Hampshire
Alaska or Hawaii
call 603–884–6660

In Puerto Rico
call 809–754–7575
x2012

ELECTRONIC ORDERS (U.S. ONLY)

Dial 800–DEC–DEMO with any VT100 or VT200
compatible terminal and a 1200 baud modem.
If you need assistance, call 1–800–DIGITAL.

DIRECT MAIL ORDERS (U.S. and Puerto Rico*)

DIGITAL EQUIPMENT CORPORATION
P.O. Box CS2008
Nashua, New Hampshire 03061

DIRECT MAIL ORDERS (Canada)

DIGITAL EQUIPMENT OF CANADA LTD.
940 Belfast Road
Ottawa, Ontario, Canada K1G 4C2
Attn: A&SG Business Manager

INTERNATIONAL

DIGITAL
EQUIPMENT CORPORATION
A&SG Business Manager
c/o Digital's local subsidiary
or approved distributor

Internal orders should be placed through the Software Distribution Center (SDC),
Digital Equipment Corporation, Westminster, Massachusetts 01473

*Any prepaid order from Puerto Rico must be placed
with the Local Digital Subsidiary:
809–754–7575

DECnet–ULTRIX
Release Notes
AA–JD82E–TE

READER'S COMMENTS

What do you think of this manual? Your comments and suggestions will help us to improve the quality and usefulness of our publications.

Please rate this manual:

	Poor				Excellent
Accuracy	1	2	3	4	5
Readability	1	2	3	4	5
Examples	1	2	3	4	5
Organization	1	2	3	4	5
Completeness	1	2	3	4	5

Did you find errors in this manual? If so, please specify the error(s) and page number(s).

_____

_____

_____

_____

General comments:

_____

_____

_____

_____

Suggestions for improvement:

_____

_____

_____

_____

Name _____ Date _____

Title _____ Department _____

Company _____ Street _____

City _____ State/Country _____ Zip _____

DO NOT CUT – FOLD HERE AND TAPE

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY LABEL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

digital™

Networks and
Communications Publications
550 King Street
Littleton, MA 01460–1289

DO NOT CUT – FOLD HERE

CUT ON THIS LINE

digital

DECnet–ULTRIX

Installation

May 1990

This manual shows you step by step how to install your DECnet–ULTRIX software and how to configure and test your node's operation in the network.

Supersession/Update Information:	This is a revised manual.
Operating System and Version:	ULTRIX V4.0
Software Version:	DECnet–ULTRIX V4.0

digital™

Order Number: AA–EA87E–TE

AA–EA87E–TE
May 1990

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by Digital or its affiliated companies.

Restricted Rights: Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227–7013.

The following are trademarks of Digital Equipment Corporation:

DEC	PDP	VAX
DECnet	ULTRIX	VMS
DECUS	UNIBUS	digital™

UNIX is a registered trademark of AT&T in the USA and other countries.

This manual was produced by Networks and Communications Publications.

Contents

Tables

Preface

This manual explains how to install DECnet–ULTRIX V4.0 software on any ULTRIX V4.0 system.

Using this manual, you can determine which type of installation procedure to use and which software subsets to install. The examples lead you step by step through the installation procedures.

This manual also tells you how to configure both the server and the system to run the optional DECnet–Internet Gateway software.

Finally, you learn how to verify that your node is running properly on the network.

Intended Audience

This manual is for the user who is either installing DECnet–ULTRIX software onto a running ULTRIX V4.0 system or testing a new DECnet–ULTRIX node in the network. Ideally, the person who installs the software has had system administration experience. Be aware that most of the procedures described in this manual require superuser privileges.

Structure of This Manual

This manual contains four chapters and an appendix:

Chapter 1	Reviews the four questions you must answer before you begin the DECnet–ULTRIX installation: Has DECnet been configured into the kernel yet? Have you used LMF to register your DECnet–ULTRIX V4.0 software license? Are you installing DECnet–ULTRIX onto a node or into a diskless environment? Which software subsets do you want to install?
Chapter 2	Explains how to install DECnet–ULTRIX software onto a node.
Chapter 3	Explains how to install DECnet–ULTRIX for a client to use in a diskless environment.
Chapter 4	Describes how to check the node's operation on the network.
Appendix A	Lists all files that the DECnet–ULTRIX installation procedure copies to your system.

Related Documents

For more information about DECnet-ULTRIX software, see the following documents:

- *DECnet–ULTRIX Release Notes*

  This document contains miscellaneous information and updates not included in other books in the DECnet–ULTRIX documentation set.

- *DECnet–ULTRIX DECnet–Internet Gateway Use and Management*

  This manual describes how to use and manage the DECnet–Internet Gateway.

- *DECnet–ULTRIX Programming*

  This manual explains application programming concepts and guidelines to use in the DECnet–ULTRIX environment. The manual also describes DECnet–ULTRIX system calls and subroutines and shows DECnet–ULTRIX data structures and programming examples.

- *DECnet–ULTRIX Network Management*

  This manual describes procedures for managing the network, such as defining the permanent and volatile databases, node identifications and addresses, and lines and circuits; enabling event logging; displaying network counter information; operating and controlling a DECnet–ULTRIX node; and testing the network operation.

- *DECnet–ULTRIX NCP Command Reference*

  This manual tells network managers how to use the Network Control Program (**ncp**)to perform network management functions.

To obtain a detailed description of the Digital Network Architecture, see *DECnet Digital Network Architecture (Phase IV), General Description*.

For a detailed description of the License Management Facility (LMF), see *Guide to Software Licensing*.

For a beginner's introduction to the ULTRIX operating system, see *The Little Gray Book: An ULTRIX Primer*.

Graphic Conventions

This manual uses the following conventions:

- All numbers are decimal unless otherwise noted.
- All Ethernet addresses are hexadecimal.

Convention	Meaning
special	In running text, commands, command options, user names, file names, and directory names appear in **special** type.
`example`	Indicates an example of system output or user input. System output is in black type; user input is in red type.

Convention	Meaning
lowercase/UPPERCASE	Because the ULTRIX software is case sensitive, type all literal input in the case shown. In running text, UPPERCASE is also used for the names of all DECnet nodes, including DECnet–ULTRIX nodes. This convention follows DECnet protocol, which names and recognizes all nodes in UPPERCASE. However, node names are not case sensitive and need not be typed in the case shown.
italic	Indicates a variable, for which either you or the system specifies a value.
[ ]	Within the installation procedure, indicates the default value. Do not type the square brackets.
KEY	Indicates a key on your keyboard. CTRL/KEY represents a CONTROL key sequence, where you press the CONTROL key at the same time as the specified key. Note that keyboard keys are represented by this symbol, <*key*>, on line.
RET	Indicates the RETURN key.

Chapter 1

Before You Start

Using this manual, you can install DECnet–ULTRIX V4.0 software onto VAX or RISC systems. There are only three differences between the two types of installations:

- The ULTRIX and DECnet–ULTRIX software subset names begin with a three-letter prefix, which indicates whether the subset is for a VAX or RISC system, as follows:

System Type	DECnet-ULTRIX Subset Names Prefix	ULTRIX Subset Names Prefix
VAX	DNU	ULT
RISC	DNP	UDT

- The DECnet–ULTRIX configuration files are named according to the type of system you have, as follows:

System Type	Configuration File Name
VAX	/sys/conf/VAX/*HOSTNAME*
RISC	/sys/conf/mips/*HOSTNAME*

- The amount of memory needed to install DECnet–ULTRIX software onto VAX and RISC systems differs. (Tables 2-1 and 3-1 list the installation requirements.)

Before you start the DECnet–ULTRIX installation procedure, answer the following questions:

- Has DECnet been configured into the kernel already? See Section 1.1.
- Have you already used the License Management Facility (LMF) to register your DECnet–ULTRIX V4.0 software license? See Section 1.2.
- Which of the two types of installation procedures do you want to use: the DECnet–ULTRIX basic installation or the DECnet–ULTRIX diskless environment installation? See Section 1.3.
- Which software subsets do you want to install: the DECnet–ULTRIX Base Software, the DECnet–Internet Gateway, the DECnet–ULTRIX On-Line Manual Pages, or the DECnet–ULTRIX Unsupported Software? See Section 1.4.

Section 1.5 explains how to use this manual.

1.1 Checking Whether DECnet Is Already Configured

Before you begin the DECnet–ULTRIX installation, check to see whether DECnet has already been configured in the ULTRIX kernel. You can install and configure DECnet–ULTRIX without configuring DECnet in the kernel; however, to run DECnet–ULTRIX, DECnet must be so configured.

The installation procedure automatically checks whether DECnet is already configured in the kernel. If DECnet is not configured, a message tells you to rebuild the kernel after the configuration is complete.

To see whether DECnet is already configured into the kernel before you start the DECnet–ULTRIX installation, enter this command:

```
% nm /vmunix | fgrep -s nsp_usrreq && echo yes RET
```

If the system displays **yes**, then DECnet is already configured. If not, configure DECnet in the kernel. You can rebuild the kernel now or after the installation and configuration are complete. (To add DECnet to the kernel, specify "options DECNET" and "pseudo-device decnet" in the appropriate configuration file and rebuild the kernel.)

1.2 Registering Your Software Before Installation

Use the ULTRIX LMF utility to register your software license for DECnet–ULTRIX V4.0. You must register your software license before you can perform any remote access. Use the information from your Product Authorization Key (PAK) to register your software.

Register and load your authorization key before you begin the DECnet–ULTRIX installation procedure. If you do not register beforehand the installation procedure reminds you to do so as soon as the installation is complete.

The License Management Facility (LMF) is a system management software tool that you use to comply with your license agreement. The LMF utility offers options for many kinds of license agreements. The terms and conditions of your contract determine your legal use of this software.

The LMF does the following:

- Maintains the file of registered PAKs (Product Authorization Keys)
- Updates the kernel cache
- Maintains a library of functions used by licensed software

NOTE

You need superuser privileges to use the LMF utility.

The *Guide to Software Licensing* introduces the LMF utility and describes how you use it. You can also refer to the LMF man page, **lmf(8)**, for information.

1.3 Choosing the Installation Procedure

How you plan to use DECnet-ULTRIX (on a node with a disk or in a diskless environment) determines which installation procedure you choose. In a diskless environment, diskless client nodes can use DECnet-ULTRIX.

- If you are installing DECnet–ULTRIX onto a node that has its own disk but does not support clients, use the DECnet–ULTRIX basic installation procedure described in Chapter 2.
- If you are installing DECnet–ULTRIX for a client to use in a diskless environment, use the installation procedure described in Chapter 3.
- If you are installing DECnet–ULTRIX onto a server node for the server's own use, use the basic installation procedure described in Chapter 2. You can perform this procedure even if you have already installed DECnet–ULTRIX into a diskless environment on the server.

1.4 Deciding Which Software Subsets to Install

Before you begin the installation, decide which subsets you want to install. The DECnet–ULTRIX software distribution media contains the following subsets:

- DECnet–ULTRIX Base Software
- DECnet–Internet Gateway
- DECnet–ULTRIX On-Line Manual Pages
- DECnet–ULTRIX Unsupported Software

Use the descriptions in Table 1-1 to determine which subsets to install. Tables 2-1 and 3-1 describe the software installation requirements for each subset.

Table 1–1: Summary of the DECnet–ULTRIX Software Subsets

Subset	DECnet–ULTRIX Base Software	DECnet–Internet Gateway	DECnet–ULTRIX On-Line Manual Pages	DECnet–ULTRIX Unsupported Software
Function	Provides the DECnet–ULTRIX functions that let you share information with remote DECnet users and programs over the network. Also required to install the DECnet–Internet Gateway.	Lets UNIX users on Internet networks and DECnet users exchange data. To set up your ULTRIX node to act as a DECnet–Internet Gateway, install the Gateway software.	Is an on-line reference manual. Gives the syntax, description, diagnostic messages, restrictions, and examples for DECnet–ULTRIX commands, system calls, and subroutines.	Contains utilities and sample programs that Digital Equipment Corporation supplies on an "as-is" basis for general customer use. However, Digital neither supports these utilities nor covers them under any Digital support contracts.
Subset For VAX Systems	DNUBASE400	DNUINETGW400	DNUMAN400	DNUUNS400
Subset For RISC Systems	DNPBASE400	DNPINETGW400	DNPMAN400	DNPUNS400

1.5 How to Use This Manual

This manual discusses the two kinds of installations you can perform in a separate chapter. Chapter 2 guides you through a DECnet–ULTRIX basic installation, and Chapter 3, a DECnet–ULTRIX diskless installation. Before you start an installation, make sure you meet the requirements shown at the beginning of the respective chapter. To check your progress during installation, see the examples in the chapter. For a complete list of the files that the installation procedure copies to your system, see Appendix A.

To make sure your node is operating on the network, follow the test procedures described in Chapter 4. The checkout procedures verify that your node can communicate with itself and other nodes on the network.

Chapter 2

Installing DECnet–ULTRIX onto a Node

This chapter tells you how to install DECnet–ULTRIX onto a node, including:

- What to do first
- How to install the software
- How to delete the installed software
- How to restart the installation
- How to correct installation errors
- How to verify the installation

Before you start the installation, make sure you meet the requirements outlined in the next section.

2.1 DECnet–ULTRIX Basic Installation Requirements

For any DECnet–ULTRIX installation, you need superuser privileges. To install DECnet–ULTRIX onto a node, you must also meet the requirements in Table 2-1.

Table 2–1: DECnet–ULTRIX Basic Installation Requirements

Software Subset	Requirements	
	For VAX Systems	**For RISC Systems**
DECnet–ULTRIX Base Software		
DECnet–ULTRIX V4.0 Distribution media or network kit	1 1600-bpi nine-track magnetic tape, CDROM, or 1 TK50 cartridge	1 1600-bpi nine-track magnetic tape, CDROM, or 1 TK50 cartridge
Operating system	ULTRIX V4.0	ULTRIX V4.0
Installed software	ULTRIX V4.0 Base Software (ULTBASE400)	ULTRIX V4.0 Base Software (UDTBASE400)
	ULTRIX V4.0 Kernel Configuration Files (ULTBIN400)	ULTRIX V4.0 Kernel Configuration Files (UDTBIN400)
Privileges	Superuser	Superuser
Disk space during installation	1750 kilobytes (KB) in /usr	2730 KB in /usr
	+ 15 KB in the root file system	+ 15 KB in the root file system
Disk space after installation	Same as during installation	Same as during installation
Storage device on a MicroVAX	1 RD53 disk or equivalent	
Estimated time	10 to 30 minutes	10 to 30 minutes
DECnet-Internet Gateway	***Additional Requirements***	
Installed software	DECnet–ULTRIX V4.0 Base Software (DNUBASE400)	DECnet–ULTRIX V4.0 Base Software (DNPBASE400)
	ULTRIX V4.0 Communications Utilities (ULTCOMM400)	ULTRIX V4.0 Communications Utilities (UDTCOMM400)
Disk space	205 KB in /usr	340 KB in /usr
Estimated time	5 to 10 minutes	5 to 10 minutes
DECnet On-Line Manual Pages	***Additional Requirements***	
Installed software	DECnet–ULTRIX V4.0 Base Software (DNUBASE400)	DECnet–ULTRIX V4.0 Base Software (DNPBASE400)
	ULTRIX V4.0 On-Line Manual Pages (ULTMAN400)	ULTRIX V4.0 On-Line Manual Pages (UDTMAN400)
Disk space	135 KB in /usr/man	135 KB in /usr/man
Estimated time	5 minutes	5 minutes
DECnet-ULTRIX Unsupported Software	***Additional Requirements***	
Installed software	DECnet–ULTRIX V4.0 Base Software (DNUBASE400)	DECnet–ULTRIX V4.0 Base Software (DNPBASE400)
Disk space	370 KB in /usr	535 KB in /usr
Estimated time	5 minutes	5 minutes

Before you install DECnet–ULTRIX:

1. Make sure you have enough free disk space to install the software subsets of your choice. To check the available disk space in **/usr** and **root** (/), enter the commands:

   ```
   # df /usr RET
   # df / RET
   ```

2. Make a backup copy of your system disk.

3. If you are using the distribution media to install the software, make sure that you have a complete distribution kit. Each DECnet–ULTRIX distribution kit consists of one or more volumes of software and a set of documentation. The Bill of Materials (BOM) included in the kit specifies its contents. Compare the items in the kit against those listed in the BOM.

   If you are using an Internet network kit to install DECnet–ULTRIX, see the *ULTRIX Remote Installation Service* for requirements. The Internet network kit lets you install the software subsets on MicroVAX, VAXstation, and DECstation 3100 products over the network.

4. Make sure you have a guest account in your system password file. The guest account enables you to use DECnet–ULTRIX features without supplying access control information. The default user account for DECnet-supplied objects and the Gateway is defined as **guest**.

   To define a guest account, log in as superuser and enter the **adduser** command. When the command prompts you for the login name of the new user, enter the name **guest** in lowercase. For example:

   ```
   # adduser RET
   Enter login name for new user (initials, first or last
   name): guest RET
   ```

   The **adduser** command then prompts you for information about the new user. For further instructions on using **adduser** to create a new account, see the *Guide to System Environment Setup* in your ULTRIX documentation set.

   After setting up the guest account, use the ULTRIX **vipw** command to enter the string NoLogin in the password field. Otherwise, you will have a **guest** account with no password.

5. Check if you are currently running DECnet. To do so, enter this command:

   ```
   # ncp show executor RET
   ```

   If the system display includes "State = On," then DECnet is running. Turn DECnet off by entering the following command:

   ```
   # ncp set executor state off RET
   ```

6. The DECnet installation procedure uses one of the following configuration files:

   - **/sys/conf/vax/*HOSTNAME***, if you are installing DECnet–ULTRIX onto a VAX system.
   - **/sys/conf/mips/*HOSTNAME***, if you are installing DECnet–ULTRIX onto a RISC system.

   For the variable *HOSTNAME*, enter the name of your host in uppercase letters. Make sure that the communication device names and other information in this configuration file are current.

7. Make sure that you know this information about your node:
   - Node name
   - Node address
   - DECnet node identification
   - Device name

2.2 Installing or Reinstalling DECnet–ULTRIX

You can use either the DECnet–ULTRIX software distribution media or an Internet network kit to install DECnet–ULTRIX onto a node. Refer to the *ULTRIX Remote Installation Service* and **ris(8)** in the *ULTRIX Reference Pages, Section 8*, for information about setting up a network kit.

To start the installation script, first use **cd** to change the working directory to root. Then enter the ULTRIX **setld(8)** (set load) command.

The **setld** command requires different options, depending on whether you are installing the software for the first time or reinstalling it. Tables 2-2 and 2-3 summarize these options, and the next sections explain them in more detail.

Table 2–2: Commands to Install DECnet–ULTRIX for the First Time

From the distribution media:	**setld -l /dev/***device*
From a network kit:	**setld -l** *hostname*:
From disk or CDROM:	**setld -l** */mount_point*

Table 2–3: Commands to Reinstall DECnet–ULTRIX

From the distribution media:	**setld -d** *subset* [...]
	setld -l /dev/*device subset* [...]
From a network kit:	**setld -d** *subset* [...]
	setld -l *hostname*: *subset* [...]
From disk or CDROM:	**setld -d** *subset* [...]
	setld -l */mount_point subset* [...]

2.2.1 Installing DECnet–ULTRIX for the First Time

- If you are using the distribution media to install the software for the first time, use the following **setld** format:

  setld -l /dev/*device*

  where

  device is the name of the device where you mount the distribution media.

  For example:

```
# cd / RET
# /etc/setld -l /dev/rmt0h RET
```

- If you are using a network kit over an Internet network to install the software for the first time, use this **setld** format:

  setld -l *hostname*:

  where

  hostname is the name of the host from which you are loading the software.

2.2.2 Reinstalling DECnet–ULTRIX

If you are reinstalling DECnet–ULTRIX, use **setld** with the **-d** option to delete the product before installing it again. Then, restart the installation and specify the DECnet–ULTRIX subsets you wish to reinstall using **setld** with the **-l** option.

For a more detailed explanation on how to use **setld** with the **-d** and **-l** options, see Sections 2.8 and 2.9.

For more information about **setld(8)**, see the *ULTRIX Guide to System Environment Setup*.

2.3 Mounting the Distribution Media

If you are installing DECnet–ULTRIX from the distribution media, the installation script tells you to make sure the media is mounted. The script then asks if you are ready. When you have mounted the media, type **y** (yes) to answer the question.

If you are installing the software from a CDROM disk, specify the mount point of the disk. The DECnet kit is located in the **c** partition under the **/VAX/DECNET** and **/RISC/DECNET** subdirectories. Mount this partition before starting the installation. For example, if your CDROM is in drive **rz1** and the mount point **/mnt** is free, you would enter the following commands:

- If you are installing DECnet–ULTRIX onto a VAX system:

```
# mount /dev/rz1c /mnt RET
# setld -l /mnt/VAX/DECNET RET
```

- If you are installing DECnet–ULTRIX onto a RISC system:

```
# mount /dev/rz1c /mnt RET
# setld -l /mnt/RISC/DECNET RET
```

2.4 Selecting the Software Subsets

If you are reinstalling DECnet–ULTRIX, skip this section and go to Section 2.5.

If you are installing DECnet–ULTRIX for the first time, the script asks you to choose the software subsets that you want to install. Select the DECnet–ULTRIX Base Software and any of the optional subsets:

```
*** ENTER SUBSET SELECTIONS ***

The subsets listed below are optional:

1) DECnet-ULTRIX Base Software          2) DECnet-Internet Gateway
3) DECnet On-Line Manual Pages          4) DECnet-ULTRIX Unsupported
                                           Software
5) All of the above
6) None of the above
7) Exit without installing subsets

Enter your choice(s):
```

Type the numbers of the options that you want to install. If you type more than one number, separate each number with a space, not a comma. Next, the script allows you to verify your choice. For example:

```
You are installing the following subsets:
DECnet-ULTRIX Base Software             DECnet-Internet Gateway
DECnet On-line Manual Pages             DECnet-ULTRIX Unsupported
                                           Software

Is this correct (y/n)?
```

If you chose the wrong options, type **n** to indicate that the subsets are not correct; the subset menu redisplays, and you can reselect your subsets. If you chose the correct options, type **y**.

2.5 Copying the DECnet–ULTRIX Software

If you have not turned off DECnet, the installation script turns it off. The system copies the software subsets to your disk and verifies them. If you are installing more than one subset, the system copies and verifies the Base Software before the Gateway or DECnet On-line Manual Pages software.

In this example, the script displays the following messages as it copies all the software subsets onto a VAX system:

```
Copying DECnet-ULTRIX Base Software (DNUBASE400) from media
Verifying DECnet-ULTRIX Base Software (DNUBASE400)

Copying DECnet-Internet Gateway (DNUINETGW400) from media
Verifying DECnet-Internet Gateway (DNUINETGW400)

Copying DECnet On-line Manual Pages (DNUMAN400) from media
Verifying DECnet On-line Manual Pages (DNUMAN400)

Copying DECnet-ULTRIX Unsupported Software (DNUUNS400) from media
Verifying DECnet-ULTRIX Unsupported Software (DNUUNS400)
```

NOTE

If you are installing DECnet–ULTRIX onto a RISC system, the prefix DNP, rather than DNU, appears in the subsets.

If verification fails, look in **/usr/var/adm/fverifylog** and **/etc/setldlog** for information to help you correct the error. Make the correction and restart the installation by using the **setld** command with the **-l** option. See Section 2.9 for details.

2.6 Configuring the Node for DECnet–ULTRIX Base Software

To configure the node to run the DECnet–ULTRIX base software, the script prompts you for your node name, node address, DECnet node identification, and device name. If you need help responding to any of these prompts, type a question mark (?) and press RET.

If you type the wrong answer for any question, you can change it by typing n when asked to verify your answer. The procedure then redisplays the question so that you can enter the correct information.

The script begins the configuration by displaying these informational messages:

```
Configuring the node to run DECnet-ULTRIX V4.0 base software.

You will be asked a few questions during the DECnet-ULTRIX V4.0
configuration.

If you need more information to answer a question, you can type ?
at the prompts, or consult the DECnet-ULTRIX Installation manual.

The DECnet library routines are now located in their own
library, -ldnet.
```

If you have not registered your DECnet–ULTRIX license, the script displays this message:

```
*****************    W A R N I N G    ********************
*                                                         *
* You have NOT registered the DECnet-ULTRIX License !!    *
* You will not be able to perform any remote access       *
* through DECnet-ULTRIX.                                  *
*                                                         *
* In order to do remote access, use the lmf utility to    *
* register a licensed DECnet-ULTRIX PAK.                  *
*                                                         *
***********************************************************

[ Press the RETURN key to continue :]
```

This message is simply a reminder that you must register your DECnet–ULTRIX license after the installation and configuration are complete. You do not have to interrupt the configuration at this point.

The script then displays question 1:

```
The next question confirms that you are ready to configure your
system to run DECnet at this time.  To configure DECnet you need to
know what your DECnet node name and address are.  Also, if you want
to configure your DECnet nodes database during this configuration
process, you need to know the DECnet names and addresses of those
nodes you wish to define.

If you answer n (no) to this question now, you are told how
you can configure DECnet later.

1. Do you want to configure your system to run DECnet? (y/n) [y]:
```

To answer "yes," press RET. The script moves on to question 2.

If you do not wish to configure the system to run DECnet, answer "no" by typing n. The following message appears before you automatically exit the script:

```
You chose not to configure your system to run DECnet at this time.
If you decide to configure DECnet later, issue the command:

        setld -c DNUBASE400 INSTALL
```

If you are installing DECnet–ULTRIX onto a RISC system, the installation displays this command instead:

```
setld -c DNPBASE400 INSTALL
```

With question 2, the script prompts you for your DECnet node name.

```
2. What is your DECnet node name (1-6 chars)? []: node

   Your node name is node

   Is this correct? (y/n) [n]:
```

Enter a node name of 1 to 6 alphanumeric characters, including at least one alphabetic character. If possible, use the same node name that you specified as your Internet host name when you installed ULTRIX. Consult your DECnet network manager before choosing your node name and address. Respond by typing a y or n, as appropriate.

```
3. What is your DECnet node address (aa.nnnn)? []: aa.nnnn RET

   Your node address is aa.nnnn

   Is this correct? (y/n) [n]:
```

Your DECnet node address consists of an area number, a period, and a node number. The area number is a decimal integer from 1 to 63; the node number is a decimal number from 1 to 1023. Note that this address is different from your Internet host address. Respond by typing a y or n, as appropriate.

```
4. What is your DECnet Node Identification -
   string? (1-32 chars) []: identification string RET

   Your node identification string is

   identification string

   Is this correct? (y/n) [n]:
```

Enter a string of 1 to 32 ASCII characters. You can include blanks but not shell metacharacters (! ? ' " *). The node identification string is a short message that appears next to the node ID in network management displays. Its purpose is to establish the node's identity on the network by indicating its type, use, users, or some other distinguishing feature. Respond by typing a y or n, as appropriate.

```
5. What is the name of the device on which you will be
   running the DECnet software (dev-c)? []: dev-c RET
```

The script asks question 5 only if it finds more than one DECnet-supported network device in the system configuration file. The name of the configuration file depends on the type of system (VAX or RISC) you have, as follows:

System Type	Configuration File Name
VAX	/sys/conf/vax/*HOSTNAME*
RISC	/sys/conf/mips/*HOSTNAME*

The variable *HOSTNAME* is the host name of your system in uppercase letters. The device name consists of three alphabetic characters, a dash, and an integer from 0 to 65535. Following is a list of the possible DECnet device names and ULTRIX equivalents:

DECnet Device Name	Equivalent ULTRIX Device Name
una-*n*	de*n*
qna-*n*	qe*n*
dmc-*n*	dmc*n*
dmv-*n*	dmv*n*
sva-*n*	ln*n*
bnt-*n*	ni*n*
xna-*n*	xna*n*

2.6.1 Configuring the DECnet Nodes Database

Each node has a DECnet nodes database that may contain the following information:

- The DECnet names and addresses for the nodes on your DECnet network.
- The names, addresses, and down-line loading information for clients that you add to a diskless environment.

If any nodes or diskless clients are already defined in your database, the installation procedure keeps the database and displays this message:

```
A nodes database has been found on your system and will be retained.
If you wish to define new nodes, use the ncp define node command after
the installation is complete.
```

If the database does not exist, the installation procedure gives you the option of defining some DECnet nodes in the permanent nodes database. This message appears:

```
This part of the installation procedure builds your DECnet nodes database.
Questions 6 and 7 prompt you for a DECnet node and address for each node
that you want to define in the database.

Questions 6 and 7 will reoccur until you type <RET> at question 6.
```

If you plan to copy an existing database from another node, define that node when the script prompts you with questions 6 and 7. You add the node to your database so you can access it later to copy its nodes database. After the installation, you can use the **update_nodes -f** *node* command to copy the database. (The **update_nodes** command is an unsupported utility in the DECnet–ULTRIX Unsupported Software subset and documented in the *DECnet–ULTRIX Release Notes*.)

Questions 6 and 7 appear as follows:

```
6. Enter node name (1-6 chars or <RET> to continue): node RET
```

Enter a node name of 1 to 6 alphanumeric characters, including at least one alphabetic character.

```
7. Enter node's node address (aa.nnnn): aa.nnnn RET
```

The DECnet node address consists of an area number, a period, and a node number. The area number is a decimal integer from 1 to 63; the node number is a decimal number from 1 to 1023. Note that this address is different from the Internet host address.

If you enter a node name or address incorrectly, just reenter both the node name and address when the questions redisplay. The script deletes the error for you.

When you have finished entering the nodes in the database, press RET in response to question 6.

After installation, you can define nodes in the permanent database by using the **ncp define node** command. Also, you can delete node names and addresses by using the **ncp purge node** command. For more information, see the *DECnet–ULTRIX Network Management* manual and the *DECnet–ULTRIX NCP Command Reference*.

2.6.2 Modifying System Files

The installation procedure then modifies your system files. This generally takes no longer than 10 minutes.

Before the installation procedure edits any files, it saves each one in a file of the name:

```
filename.savn
```

where *n* is a version number that is incremented on each installation.

As the script modifies the files, it displays the following messages, as appropriate:

```
Modifying rc.local
Creating DECnet proxy file, /etc/dnet_proxy
Initializing DECnet database

After verifying that the system is operating properly with DECnet
installed you may wish to remove the following saved file:

   - /etc/rc.local.sav[n]
```

If a proxy file already exists on your system, the script does not display the message about creating a proxy file. For information about creating a proxy file, see the *DECnet–ULTRIX Network Management* manual.

If you have edited any system file to customize it prior to this installation, check the new file to make sure the modifications are compatible with your previous edits.

2.6.3 Preparing to Restart the Node

At this time, the installation procedure performs several checks and displays information messages as needed. Before you start DECnet on the node, you may need to make some final changes to your environment. For example:

- If DECnet is not already configured into your ULTRIX kernel, the script displays this message:

  ```
  DECnet is not built into the currently running kernel. After
  configuration, you should rebuild your kernel with DECnet and
  then reboot to complete the installation.
  ```

  This message is simply a reminder that you must rebuild the kernel when the configuration is complete. Do not interrupt the configuration to rebuild the kernel.

  To add DECnet to the kernel, specify "options DECNET" and "pseudo-device decnet" in the appropriate configuration file and rebuild the kernel.

- If DECnet has already been configured into your kernel, the script reminds you to start DECnet:

  ```
  8. Do you want to start DECnet now (y/n)? [y]:
  ```

Press RET or type **y** to answer "yes". If you answer "yes", the script automatically starts DECnet for you. If you answer "no", the script displays this message:

```
You can turn DECnet on manually by rebooting your system
or by entering the following command:

/usr/bin/ncp set exec state on
```

- If you have a lat control program (lcp) command line in **/etc/rc.local**, the script displays this message:

```
A lat control program (lcp) command line was found in
/etc/rc.local.  When this procedure is finished, you should
edit rc.local and move the lcp line(s) to follow the ncp command.
If the lcp command precedes the ncp command, your lat terminal
lines may not work properly.
```

- If you do not have a guest account in your system, the script displays this message:

```
There is no guest account in your system.  If you wish to access
this system from remote systems, you should have a guest account
specified.  A guest account is needed for the proper operation of
dlogind, dtermd, and dtr.

You should set a password for this account to prevent unauthorized
access to your system.
```

  If you need instructions on how to create a new account, see the *ULTRIX Guide to System Environment Setup*.

- If you have not already registered your software license, the script displays this message:

```
Don't forget to register your DECnet-ULTRIX PAK
(Product Authorization Key) using lmf.  Otherwise
you won't be able to access remote systems.

After you register your DECnet-ULTRIX PAK, you must
either reboot or issue the following commands:

/usr/etc/lmf reset
/usr/bin/ncp set exec state on
```

When you start DECnet, your system displays the following message:

```
Starting DECnet
```

Your system comes up with the executor and circuit states set to **on**, as reported by the following event messages:

```
Event type 2.0, Local node state change
Occurred 12-OCT-90 16:47:30.0 on node aa.nnnn (NODE)
Operator commands
Old state = off
New state = on

Event type 4.10, Circuit up
Occurred 12-OCT-80 16:47.32.0 on node aa.nnnn (NODE)
Circuit DEV-C
Adjacent node = aa.nnnn (NODE)
```

The default logging device for all events is the console. For instructions on how to disable logging or redirect it to another device, see the *DECnet–ULTRIX Network Management* manual.

Once your DECnet–ULTRIX node is running, you can log in and use the checkout procedure as described in Chapter 4. This procedure tests your node's operation in the network.

2.7 Configuring the Node to Run Gateway Software

If you chose to install the DECnet–Internet Gateway subset, the script now configures the node to run the DECnet–Internet Gateway software.

The script begins with question 1, which reads as follows:

```
The next question confirms that you are ready to configure your system
to run the DECnet-Internet Gateway at this time.  In order to run the
Gateway you must have already configured your system to run DECnet.

The configuration procedure asks no questions.  If you decide not to
configure the Gateway at this time you will be told how to configure
it at a later date.

1. Do you want to configure your system to run the
   DECnet-Internet Gateway? (y/n) [y]:
```

To answer "yes," press RET. The script stops asking questions.

If you do not wish to configure the system to run the DECnet–Internet Gateway, answer "no" to question 1. The following message appears before you automatically exit the script:

```
You chose not to configure your system to run the Gateway at this
time. If you decide to configure the Gateway at a later date, you
can issue the command:

        setld -c DNUINETGW400 INSTALL
```

If you are installing DECnet–ULTRIX software onto a RISC system, the installation displays this command instead:

```
        setld -c DNPINETGW400 INSTALL
```

If you answer "yes" to question 1, the installation script edits the **/etc/inetd.conf** file and displays this message:

```
The file /etc/inetd.conf will be edited to use
/usr/etc/ftpd.gw and /usr/etc/telnetd.gw instead
of /usr/etc/ftpd and /usr/etc/telnetd.

Editing /etc/inetd.conf

After verifying that the system is operating properly as a
gateway you may wish to remove the following saved file:

   - /etc/inetd.conf.sav[n]
```

NOTE

If you are installing the DECnet–ULTRIX Base Software and DECnet–Internet Gateway software for the first time, you must reboot the system to start the Gateway. If DECnet–ULTRIX was previously installed and you are installing only the Gateway, you can either kill and restart **inetd** without rebooting or send a hang-up signal to the **inetd** daemon.

The DECnet–Internet Gateway, by default, is configured as a bidirectional gateway, with access enabled in two directions: from DECnet to Internet systems, and from Internet to DECnet systems. When the installation and configuration are complete, you can configure the DECnet–Internet Gateway as a unidirectional gateway by disabling either DECnet-to-Internet or Internet-to-DECnet access. See the *DECnet–Internet Gateway Use and Management* manual for specific instructions on how to do this.

2.8 Deleting the DECnet–ULTRIX Software

To delete the DECnet–ULTRIX software from your system, log in as superuser and issue the **setld** command with the **-d** option, as follows:

setld -d *subset* [...]

where

subset is the name of the DECnet–ULTRIX software subsets that you are deleting. If you are reinstalling subsets, delete any subsets that you are reinstalling. You can include the names of one or more of the DECnet–ULTRIX subsets, as shown in the following table:

	Subset Names for *subset* Variable	
Software Subset	**For VAX Systems**	**For RISC Systems**
DECnet-ULTRIX Base Software	DNUBASE400	DNPBASE400
DECnet-Internet Gateway Software	DNUINETGW400	DNPINETGW400
DECnet On-Line Manual Pages	DNUMAN400	DNPMAN400
DECnet-ULTRIX Unsupported Software	DNUUNS400	DNPUNS400

If you select any of the optional subsets, make sure you list them before the base subset. The following command deletes the DECnet–Internet Gateway, the On-Line Manual Pages, and the Base Software from a VAX system:

```
# /etc/setld -d DNUINETGW400 DNUMAN400 DNUBASE400 RET
```

The system lists the subsets you are deleting. It then asks if you want to delete the DECnet database:

```
Do you want to delete the DECnet database from
your system (y/n)?
```

Answer **y** (yes) to remove the DECnet database from **/usr/lib/dnet**. The system then displays the following messages:

```
The nodes database (/usr/lib/dnet/nodes_p and /usr/lib/dnet/nodes_v)
has been retained for possible use by MOP.  These files can be
deleted manually if they are not needed.

Your DECnet software subset has been deleted.

You may also want to remove the following saved file:
- /etc/rc.local.sav[n]
```

Note that the files **nodes_v** and **nodes_p**, which contain down-line loading information, are not removed when you delete the DECnet database. However, you can remove them separately.

2.9 Restarting the Installation

You can restart the installation by entering **setld** with the **-l** option and specifying the DECnet–ULTRIX subsets that you want to reinstall:

setld -l /dev/*device subset* [...]

where

/dev/*device*	is the name of the distribution media from which you are loading the software.
subset	is the name of the DECnet–ULTRIX software subset that you are installing. You can include the names of one or more of the DECnet–ULTRIX subsets, as shown in the following table:

	Subset Names for *subset* Variable	
Software Subset	**For VAX Systems**	**For RISC Systems**
DECnet-ULTRIX Base Software	DNUBASE400	DNPBASE400
DECnet-Internet Gateway Software	DNUINETGW400	DNPINETGW400
DECnet On-Line Manual Pages	DNUMAN400	DNPMAN400
DECnet-ULTRIX Unsupported Software	DNUUNS400	DNPUNS400

If you install any of the optional subsets with the base software, list the base software subset first. The following command deletes the Base Software and the DECnet–Internet Gateway from a VAX system:

```
# cd / RET
# /etc/setld -l /dev/rmt0h DNUBASE400 DNUINETGW400 RET
```

Notice that when you restart an installation, the order in which you list the subsets is opposite to the order in which you list them during a deletion.

2.10 Correcting Errors After the Installation Is Complete

This section describes how to correct errors after the installation is complete. For information on how to correct minor errors during the installation, see the respective sections in this chapter that describe the steps in the installation process. All such sections contain instructions for correcting errors.

If you receive an error message when you reboot the node, look in the **/usr/lib/dnet/ncp.log** file. The only errors you are likely to find when you reboot are **ncp** errors. You will be notified of any other errors as they occur.

If the installation software detects a serious error, it displays a message on your terminal and stops the procedure. In the event that the procedure stops, look for error messages in **/etc/setldlog**. When you have corrected the problem, restart the installation as described in the next paragraph.

After correcting any installation errors, restart installation by entering the **setld** command with the **-l** option. You will reinstall only the subsets that did not install correctly the first time. For a more detailed explanation of how to use the **setld** command with the **-l** option, see Section 2.9.

2.11 Verifying the Installation

To verify that DECnet–ULTRIX has been installed correctly on your node, run the checkout procedures described in Chapter 4. These procedures test whether your node can communicate with itself and with other nodes on the network. Before you begin testing your node, complete any postinstallation tasks that the script reminded you to do. For details, refer to Section 2.6.3.

Chapter 3

Installing DECnet–ULTRIX into a Diskless Environment

If you are installing DECnet–ULTRIX software for a client to use in a diskless environment, be sure to make the following preparations:

- Designate a server node for the DECnet–ULTRIX installation.
- Make sure that the diskless environment already exists on the server node.
- Check that each registered client is licensed to use the software that the server provides.

You can add registered clients to the diskless environment either before or after installing DECnet–ULTRIX software. Each client is configured when you boot it for the first time after you install DECnet–ULTRIX software on the server.

To install DECnet–ULTRIX software into a diskless environment, run the Diskless Management Services (**dms**) utility from the server node. (You can run **dms** only on the server.) The following sections describe the full procedure.

Before starting with the installation, make sure you meet the requirements outlined in the next section.

3.1 DECnet–ULTRIX Diskless Installation Requirements

To install DECnet–ULTRIX into a diskless environment, you must meet the requirements in Table 3–1.

Table 3–1: DECnet–ULTRIX Diskless Installation Requirements

Software Subset	Requirements	
	For VAX Systems	**For RISC Systems**
DECnet–ULTRIX Base Software		
DECnet–ULTRIX V4.0 Distribution media	1 1600-bpi nine-track magnetic tape, CDROM, or 1 TK50 cartridge	1 1600-bpi nine-track magnetic tape, CDROM, or 1 TK50 cartridge
Diskless environment (OS)	ULTRIX V4.0	ULTRIX V4.0
Installed software	ULTRIX V4.0 Base Software (ULTBASE400)	ULTRIX V4.0 Base Software (UDTBASE400)
	ULTRIX V4.0 Kernel Configuration Files (ULTBIN400)	ULTRIX V4.0 Kernel Configuration Files (UDTBIN400)
Privileges	Superuser	Superuser
Disk space during installation	1750 KB in the environment	2730 KB in the environment
	+ 15 KB per client root file system	+ 15 KB per client root file system
Disk space after installation	Same as during installation	Same as during installation
Estimated time	30 minutes for base installation	30 minutes for base installation
DECnet-Internet Gateway	***Additional Requirements***	
Installed software	DECnet–ULTRIX V4.0 Base Software (DNUBASE400)	DECnet–ULTRIX V4.0 Base Software (DNPBASE400)
	ULTRIX V4.0 Communications Utilities (ULTCOMM400)	ULTRIX V4.0 Communications Utilities (UDTCOMM400)
Disk space	205 KB in **/usr**	340 KB in **/usr**
Estimated Time	5 to 10 minutes	5 to 10 minutes
DECnet-ULTRIX On-Line Manual Pages	***Additional Requirements***	
Installed software	DECnet–ULTRIX V4.0 Base Software (DNUBASE400)	DECnet–ULTRIX V4.0 Base Software (DNPBASE400)
	ULTRIX V4.0 On-Line Manual Pages (ULTMAN400)	ULTRIX V4.0 On-Line Manual Pages (UDTMAN400)
Disk space	135 KB in **/usr/man**	135 KB in **/usr/man**
Estimated time	5 minutes	5 minutes
DECnet-ULTRIX Unsupported Software	***Additional Requirements***	
Installed software	DECnet–ULTRIX V4.0 Base Software (DNUBASE400)	DECnet–ULTRIX V4.0 Base Software (DNPBASE400)
Disk space	370 KB in **/usr**	535 KB in **/usr**
Estimated time	5 minutes	5 minutes

Before you install DECnet–ULTRIX into a diskless environment on a server node:

1. Set up a diskless environment for the clients. This is equivalent to installing ULTRIX V4.0 software for the clients. See the *ULTRIX Guide to Diskless Management Services* for details.
2. Make sure you have enough free disk space to install the software subsets of your choice. To check the available disk space in the environment or the root file system, enter the **df** command. For example:

   ```
   # df /usr/var/diskless/dlenv0 RET
   # df /usr/var/diskless/dlclient0 RET
   ```

3. Make a backup copy of the diskless environment.
4. Make sure you have the complete software distribution kit. Each DECnet–ULTRIX kit consists of one or more volumes of software and a set of documentation. The Bill of Materials (BOM) included with the kit specifies its contents. Compare the items in the kit with those listed in the BOM.
5. If DECnet–ULTRIX is already installed in the diskless environment, log in to each client using that environment and turn DECnet off. Do not turn DECnet off on the server node.

3.2 Running the DECnet–ULTRIX Diskless Installation Procedure

To start the DECnet–ULTRIX diskless installation procedure, log in to the server node as superuser. Then change to the root directory (/) and start the Diskless Management Service (**dms**) utility:

```
# cd / RET
# dms RET
```

The **dms** utility displays a licensing notice, then prompts you for the superuser password. When you type the superuser password and press RET, **dms** displays a menu of its services:

```
DISKLESS MANAGEMENT SERVICES (DMS) UTILITY MENU

        a - Add Client Processor
        m - Modify Client Parameters
        r - Remove Client Processor
        l - List Registered Clients
        s - Show Products in Diskless Environments
        i - Install Software
        c - Create Diskless Area on Disk
        k - Kernel Rebuild or Copy
        e - Exit

Enter your choice:
```

NOTE

Either before or after you install DECnet–ULTRIX, define the client nodes on which you will be using DECnet–ULTRIX. The **a** option defines a client. Refer to the ULTRIX documentation for more information about defining or adding clients.

To install DECnet–ULTRIX, type the **i** option at the prompt. This menu appears:

```
1 Install Operating System to New Area
2 Add Software to Existing Area
3 Return to Previous Menu

Enter your choice:
```

Choose option **2** to add DECnet–ULTRIX to an existing diskless area that already contains an operating system. This message appears with a list of the installation directories available:

```
You have chosen to install additional software into an existing diskless
environment.  These are the available installation directories:

        1 /dlenv0/root0.vax
        2 /dlenv0/root0.mips
        3 /dlenv1/root1.mips

Enter your choice:
```

Choose the diskless environment into which you want to install the software by typing the appropriate option and pressing RET.

Next, **dms** prompts you for the name of the device special file or the mount point of the distribution media:

```
Enter the device special file name or mount point of the
distribution media, for example, /dev/rmt0h
```

If the distribution kit is on a TK50 cartridge tape or a magnetic tape, enter the device special file name.

If the kit is on a CDROM disk, specify the mount point of the disk. The DECnet–ULTRIX kit is located in the **c** partition. Mount this partition before starting the installation (if you have not already mounted the partition, press CTRL/Z to suspend the installation, type the **mount** command, and type **fg** to resume the installation). For example, if your CDROM is in drive **ra1** and the mount point **/mnt** is free, you would enter this **mount** command:

```
# mount /dev/ra1c /mnt RET
```

If diskless clients have already been added to the chosen diskless environment, the **dms** utility asks the following question:

```
The product software will automatically by propagated to every
registered client.  Is that all right? (y/n):
```

If you answer yes, **dms** copies the software to each diskless client. If you answer no, **dms** terminates the installation.

3.3 Selecting Software Subsets

The script then asks you to choose the software that you want to install: the DECnet–ULTRIX Base Software, the DECnet–Internet Gateway software, the DECnet On-Line Manual Pages, the DECnet–ULTRIX Unsupported Software, or all four subsets:

```
*** ENTER SUBSET SELECTIONS ***

The subsets listed below are optional.

1) DECnet-ULTRIX Base Software          2) DECnet-Internet Gateway
3) DECnet On-Line Manual Pages          4) DECnet-ULTRIX Unsupported
                                           Software

5) All of the Above
6) None of the Above
7) Exit without installing subsets

Enter your choice(s):
```

Type the number of the option that you want to install. If you type more than one number, separate each number with a space, not a comma. Next, the script allows you to verify your choice. For example:

```
You are installing the following subsets:

DECnet-ULTRIX Base Software             DECnet-Internet Gateway
DECnet On-Line Manual Pages             DECnet-ULTRIX Unsupported
                                          Software

Is this correct (y/n)?
```

If you chose the wrong option, type **n** to indicate that the subsets are not correct. The subset menu redisplays, and you can reselect your subsets. If you chose the correct option, type **y**.

The installation script then copies and verifies the software subsets. If you are installing all subsets, it copies and verifies the base software, then the other subsets.

In this example, the script displays the following messages as it copies all the software subsets onto a VAX system:

```
Copying DECnet-ULTRIX Base Software (DNUBASE400) from media
Verifying DECnet-ULTRIX Base Software (DNUBASE400)

Copying DECnet On-Line Manual Pages (DNUMAN400) from media
Verifying DECnet On-Line Manual Pages (DNUMAN400)

Copying DECnet-Internet Gateway (DNUINETGW400) from media
Verifying DECnet-Internet Gateway (DNUINETGW400)

Copying DECnet-ULTRIX Unsupported Software (DNUUNS400) from media
Verifying DECnet-ULTRIX Unsupported Software (DNUUNS400)
```

NOTE

If you are installing DECnet–ULTRIX onto a RISC system, the prefix DNP, rather than DNU, appears in the subsets.

If verification fails, look in the diskless environment's **/usr/var/adm/fverify** log file for information to help you correct the error. Make the correction and restart the installation by using the **dms** utility. See Section 3.2 for details.

3.4 Configuring the Client for DECnet–ULTRIX Base Software

Clients are configured when you boot them for the first time after you install DECnet–ULTRIX software on the server. This section describes the configuration script.

If you type the wrong answer for any question, you can change it by typing **n** when asked to verify your answer. The procedure then redisplays the question so that you can enter the correct information. If you need help responding to any of these prompts, type a question mark (?) and press RET.

The script begins the configuration by displaying some informational messages:

```
Configuring the node to run DECnet-ULTRIX V4.0 base software.

You will be asked a few questions during the DECnet-ULTRIX V4.0
configuration.

If you need more information to answer a question, you can type ?
at the prompts, or consult the DECnet-ULTRIX Installation manual.

The DECnet library routines are now located in their own
library, -ldnet.
```

If you have not registered your DECnet–ULTRIX license, the script displays this message:

```
*****************    W A R N I N G    ********************
*                                                        *
* You have NOT registered the DECnet-ULTRIX License !!   *
* You will not be able to perform any remote access      *
* through DECnet-ULTRIX.                                 *
*                                                        *
* In order to do remote access, use the lmf utility to   *
* register a licensed DECnet-ULTRIX PAK.                 *
*                                                        *
**********************************************************

[ Press the RETURN key to continue :]
```

This message is simply a reminder that you must register your DECnet–ULTRIX license after the installation and configuration are complete. You do not have to interrupt the configuration at this point.

The script then displays question 1:

```
The next question confirms that you are ready to configure your
system to run DECnet at this time.  To configure DECnet you need to
know what your DECnet node name and address are.  Also, if you want
to configure your DECnet nodes database during this configuration
process, you need to know the DECnet names and addresses of those
nodes you wish to define.

If you answer "no" to this question now, you will be told how you can
configure DECnet later.

1. Do you want to configure your system to run DECnet? (y/n) [y]:
```

To answer "yes," press RET. The script moves on to question 2.

If you do not wish to configure the system to run DECnet, answer "no" by typing n. The following message appears before you automatically exit the script:

```
You chose not to configure your system to run DECnet at this time.
If you decide to configure DECnet later, issue the command:

        setld -c DNUBASE400 INSTALL
```

If you are installing DECnet–ULTRIX software onto a RISC system, the installation displays this command instead:

```
        setld -c DNUBASE400 INSTALL
```

With question 2, the script prompts you for your DECnet node name:

```
2. What is your DECnet node name (1-6 chars)? []: node RET

   Your node name is node

   Is this correct? (y/n) [n]:
```

To specify the client's DECnet node name, enter a node name of 1 to 6 alphanumeric characters, including at least one alphabetic character. If possible, use the same node name that you specified as your Internet host name. Consult your DECnet network manager before choosing your node name and address. Respond by typing a **y** or **n**, as appropriate.

```
3. What is your DECnet node address (aa.nnnn)? []: aa.nnnn RET

   Your node address is aa.nnnn

   Is this correct? (y/n) [n]:
```

Your DECnet node address consists of an area number, a period, and a node number. The area number is a decimal integer in the range of 1 to 63; the node number is a decimal number from 1 to 1023. Note that this address is different from your Internet host address. Respond by typing a **y** or **n**, as appropriate.

```
4. What is your DECnet Node Identification -
   string (1-32 chars)? []: identification string RET

   Your node identification string is

   identification string

   Is this correct? (y/n) [n]:
```

Enter a string of 1 to 32 ASCII characters. You can include blanks but not shell metacharacters (! ? ' " *). The node identification string is a short message that appears next to the node ID in network management displays. Its purpose is to establish the node's identity on the network by indicating its type, use, users, or some other distinguishing feature. Respond by typing a **y** or **n**, as appropriate.

```
5. What is the name of the device on which you will be
   running the DECnet software (dev-c)? []: dev-c RET
```

The device name consists of three alphabetic characters, a dash, and an integer from 0 to 65535. Here is a list of the possible DECnet device names and ULTRIX equivalents:

DECnet Device Names	Equivalent ULTRIX Device Names
qna-n*n*	qen*n*
dmv-n*n*	dmvn*n*
sva-n*n*	ln*n*
xna-n*n*	xna*n*

3.4.1 Configuring the DECnet Nodes Database

The root area for each client in a diskless environment contains a DECnet node database. The DECnet node database contains the DECnet names and addresses for the nodes on your DECnet network. If any nodes are already defined in your database, the installation procedure keeps the database and displays this message:

```
A nodes database has been found on your system and will be retained.
If you want to define new nodes, use the ncp define node command after
the installation is complete.
```

If the database does not exist, the installation procedure gives you the option of defining some DECnet nodes in the permanent nodes database. This message appears:

```
This part of the installation procedure builds your DECnet nodes database.
Questions 6 and 7 prompt you for a DECnet node and address for each node
that you want to define in the database.

Questions 6 and 7 will reoccur until you type <RET> at question 6.
```

If you plan to copy an existing database from another node, define that node when the script prompts you with questions 6 and 7. You add the node to your database so you can access it later to copy its nodes database. After the installation, you can use the **update_nodes -f** ***node*** command to copy the database. (The **update_nodes** command is an unsupported utility in the DECnet–ULTRIX Unsupported Software subset and documented in the *DECnet–ULTRIX Release Notes*.)

Questions 6 and 7 appear as follows:

```
6. Enter node name (1-6 chars or <RET> to continue): node RET
```

Enter a node name of 1 to 6 alphanumeric characters, including at least one alphabetic character.

```
7. Enter salem's node address (aa.nnnn): aa.nnnn RET
```

The DECnet node address consists of an area number, a period, and a node number. The area number is a decimal integer from 1 to 63; the node number is a decimal number from 1 to 1023. Note that this address is different from the Internet host address.

If you enter a node name or address incorrectly, just reenter both the node name and address when the questions redisplay. The script deletes the error for you.

When you have finished specifying the nodes in your network, press RET in response to question 6.

After installation, you can define nodes in the client's permanent database with the **ncp define node** command. Also, you can delete node names and addresses with the **ncp purge node** command. For more information, see the *DECnet–ULTRIX Network Management* manual and the *DECnet–ULTRIX NCP Command Reference*.

3.4.2 Modifying System Files

The installation procedure then modifies the client's system files. This generally takes no longer than 10 minutes.

Before the installation procedure edits any files, it saves each one in a file of the name:

```
filename.savn
```

where *n* is a version number that is incremented on each installation.

As the script modifies the files, it displays the following messages, as appropriate:

```
Modifying rc.local
Creating DECnet proxy file, /etc/dnet_proxy
Initializing DECnet database

After verifying that the system is operating properly with DECnet
installed you may wish to remove the following saved file:

   - /etc/rc.local.sav[n]
```

If a proxy file already exists on your system, the script does not display the message about creating a proxy file. For information about creating a proxy file, see the *DECnet–ULTRIX Network Management* manual.

If any of the client's system files were edited to customize them prior to this installation, check the new files to make sure the modifications are compatible with your previous edits.

3.4.3 Preparing to Start the Client

At this time, the installation procedure performs several checks and displays informational messages as needed. Before you start DECnet on the client, you may need to make some final changes to your environment. For example:

- If DECnet is not already configured into your ULTRIX kernel, the script displays the following message:

  ```
  DECnet is not built into the currently running kernel. After
  configuration, you should rebuild your kernel with DECnet and
  then reboot to complete the installation.
  ```

  This message is simply a reminder that you must rebuild the kernel when the configuration is complete. Do not interrupt the configuration to rebuild the kernel.

  To include DECnet in the kernel, specify "options DECNET" and "pseudo-device decnet" in the appropriate configuration file and rebuild the kernel.

- If DECnet has already been configured into your kernel, the following question is displayed:

  ```
  8. Do you want to start DECnet now (y/n)? [y]:
  ```

  Press RET or type **y** to answer "yes". If you answer "yes," the script automatically starts DECnet for you. If you answer "no," the script displays this message:

  ```
  You can turn DECnet on manually by rebooting your system
  or by entering the following command:

  /usr/bin/ncp set exec state on
  ```

- If you have a lat control program (lcp) command line in **/etc/rc.local**, the script displays this message:

  ```
  A lat control program (lcp) command line was found in
  /etc/rc.local.  When this procedure is finished, you should
  edit rc.local and move the lcp lines to follow the ncp command.
  If the lcp command precedes the ncp command, your lat terminal
  lines may not work properly.
  ```

- If you do not have a guest account in your system, the script displays this message:

  ```
  There is no guest account in your system.  If you wish to access
  this system from remote systems, you should have a guest account
  specified.  A guest account is needed for the proper operation of
  dlogind, dtermd, and dtr.

  You should set a password for this account to prevent unauthorized
  access to your system.
  ```

  If you need instructions on how to create a new account, see the *ULTRIX Guide to System Environment Setup*.

- If you have not already registered your software license, the script displays this message:

```
Don't forget to register your DECnet-ULTRIX PAK
(Product Authorization Key) using lmf.  Otherwise
you won't be able to access remote systems.

After you register your DECnet-ULTRIX PAK, you must
either reboot or issue the following commands:

/usr/etc/lmf reset
/usr/bin/ncp set exec state on
```

When you start DECnet, your system displays the following message:

```
Starting DECnet
```

Your system comes up with the executor and circuit states set to **on**, as reported by the following event messages:

```
Event type 2.0, Local node state change
Occurred 14-OCT-90 16:34:30.0 on node aa.nnnn (NODE)
Operator commands
Old state = off
New state = on

Event type 4.10, Circuit up
Occurred 14-OCT-90 16:34:32.0 on node aa.nnnn
Circuit DEV-C
Adjacent node = aa.nnnn (NODE)
```

The default logging device for all events is the console. For instructions on how to disable logging or redirect it to another device, see the *DECnet–ULTRIX Network Management* manual.

Once your DECnet–ULTRIX node is running, you can log in and use the checkout procedure as described in Chapter 4. This procedure tests your node's operation in the network.

3.5 Configuring the System to Run the Gateway Software

If you chose to install the DECnet–Internet Gateway subset, the script must configure each client to run the DECnet–Internet Gateway software. The clients are configured when they are booted for the first time after the DECnet–ULTRIX software has been installed on the server. The configuration script is described in this section.

The script begins with question 1, which reads as follows:

```
The next question confirms that you are ready to configure your system
to run the DECnet-Internet Gateway at this time.  In order to run the
Gateway you must have already configured your system to run DECnet.

The configuration procedure asks no questions.  If you decide not to
configure the Gateway at this time you will be told how to configure
it at a later date.

1. Do you want to configure your system to run the
   DECnet-Internet Gateway? (y/n) [y]:
```

To answer "yes," press RET. The script stops asking questions.

If you do not wish to configure the system to run the DECnet–Internet Gateway, answer "no" to question 1. The following message appears before you automatically exit the script:

```
You chose not to configure your system to run the Gateway at this
time. If you decide to configure the Gateway at a later date, you
can issue the command:

        setld -c DNUINETGW400 INSTALL
```

If you are installing DECnet–ULTRIX software onto a RISC system, the installation displays this command instead:

```
        setld -c DNPINETGW400 INSTALL
```

If you answer "yes" to question 1, the installation script edits the **/etc/inetd.conf** file and displays the following message:

```
The file /etc/inetd.conf will be edited to use
/usr/etc/ftpd.gw and /usr/etc/telnetd.gw instead
of /usr/etc/ftpd and /usr/etc/telnetd.

Editing /etc/inetd.conf

After verifying that the system is operating properly as a
gateway you may wish to remove the following saved file:

   - /etc/inetd.conf.sav[n]
```

NOTE

If you are installing the DECnet–ULTRIX Base Software and DECnet–Internet Gateway software for the first time, you must reboot the system to start the Gateway. If DECnet–ULTRIX was previously installed and you are installing only the Gateway, you can either kill and restart **inetd** without rebooting or send a hang-up signal to the **inetd** daemon.

The DECnet–Internet Gateway, by default, is configured as a bidirectional gateway, with access enabled in two directions: from DECnet to Internet systems, and from Internet to DECnet systems. When the installation and configuration are complete, you can configure the DECnet–Internet Gateway as a unidirectional gateway by disabling either DECnet-to-Internet or Internet-to-DECnet access. See the *DECnet–Internet Gateway Use and Management* manual for specific instructions on how to do this.

3.6 Correcting Errors After the Installation Is Complete

This section describes how to correct errors after the installation is complete. For information on how to correct minor errors during the installation, see the respective sections in this chapter that describe the steps in the installation process. All such sections contain instructions for correcting errors.

If you receive an error message when you reboot the client, look in the **/usr/lib/dnet/ncp.log** file for the respective system. The only errors you are likely to find when you reboot are **ncp** errors. You will be notified of any other errors as they occur during installation.

If the installation software detects a serious error, it displays a message on your terminal and stops the procedure. In the event that the procedure stops, look for error messages in **/etc/setldlog**.

After you have corrected the problem, restart the installation by running **dms**, as described in Section 3.2.

3.7 Verifying the Installation

To verify that DECnet–ULTRIX has been installed correctly on your node, run the checkout procedures described in Chapter 4. These procedures test whether your node can communicate with itself and with other nodes on the network. Before you begin testing your node, complete any postinstallation tasks that the script reminded you to do. For details, see Section 3.5.3.

Chapter 4

Checking Your Node's Operation on the Network

After installing DECnet–ULTRIX software onto your ULTRIX system, you can run a series of tests verifying that your node can communicate with itself and other nodes on the network. If you want to test the performance of an operating network, see the test procedures described in the *DECnet–ULTRIX Network Management* manual.

4.1 Overview of Checkout Tests

You can connect your node to a DECnet network using an Ethernet cable or a DDCMP point-to-point line. Figure 4-1 depicts a DDCMP point-to-point connection; Figure 4-2 depicts an Ethernet connection.

Figure 4–1: DDCMP Point-to-Point Connection

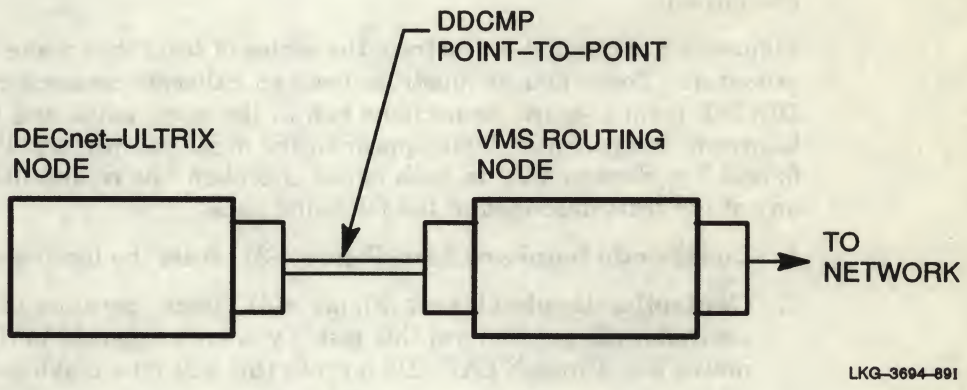
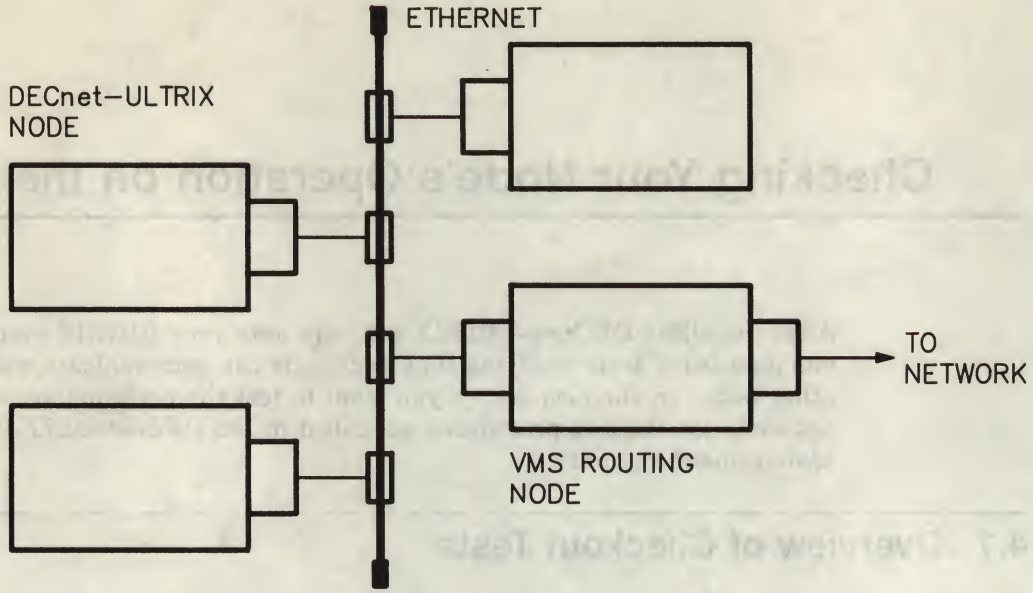


Figure 4–2: Ethernet Connection



You can use the checkout tests to verify both types of connections. If you have both Ethernet and DDCMP point-to-point hardware on your node, first run the checkout procedure on one device, then reconfigure the DECnet database and run the procedure on the second device. See the *DECnet–ULTRIX Network Management* manual for details about using **ncp** to list and define DECnet lines and circuits.

Figures 4-3 through 4-8 illustrate the series of tests that make up the checkout procedure. These figures illustrate tests on Ethernet connections; but tests on DDCMP point-to-point connections run in the same paths and test the same hardware components. Tests appear in the order you perform them (see steps 5, 6, and 7 in Section 4.2). In each figure, a broken line represents the test path for any of the tests described in the following list.

1. **Local node loopback test** (Figure 4-3). Tests the local node software.
2. **Controller loopback test** (Figure 4-4). Tests operation of the local line controller (do not perform this test if you are connected to the system over the network or through LAT). Do not use this test on a diskless client.
3. **Datalink loopback test to remote node** (Figure 4-5). Tests the physical connection to the Ethernet or DDCMP point-to-point wire and the remote system line controller and/or the datalink software.
4. **Node-level loopback test to remote node** (Figure 4-6). Tests your access to the remote network software (remote object mirror).
5. **DECnet file transfer to local node** (Figure 4-7). Tests your access to the local File Access Listener (**fal**).
6. **DECnet file transfer to remote node** (Figure 4-8). Tests your access to the remote File Access Listener (**fal**). You can use this test to check access from the Ethernet or a DDCMP point-to-point wire through a routing node to another remote node.

Figure 4–3: Local Node Loopback Test

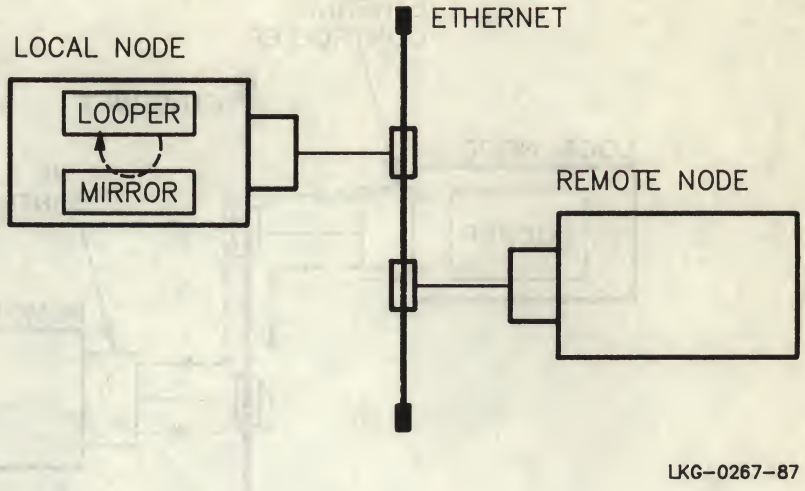


Figure 4–4: Controller Loopback Test

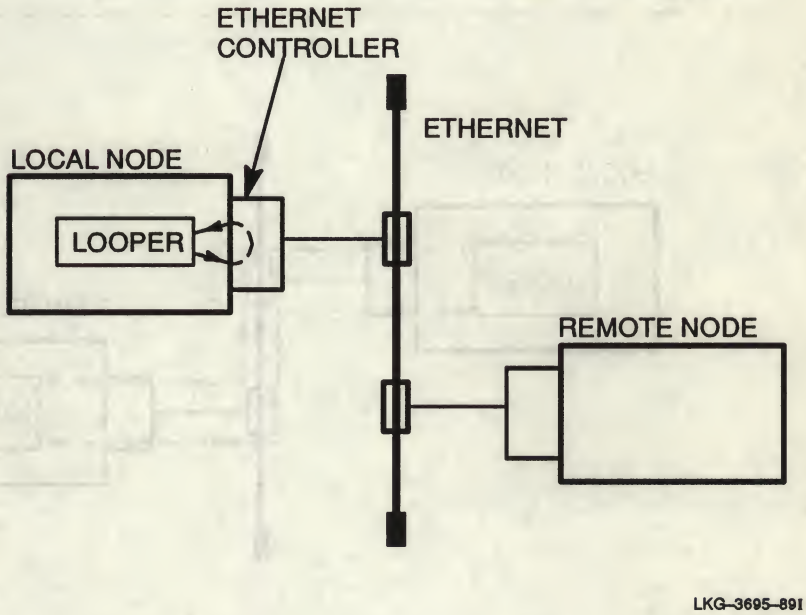


Figure 4–5: Datalink Loopback Test to Remote Node

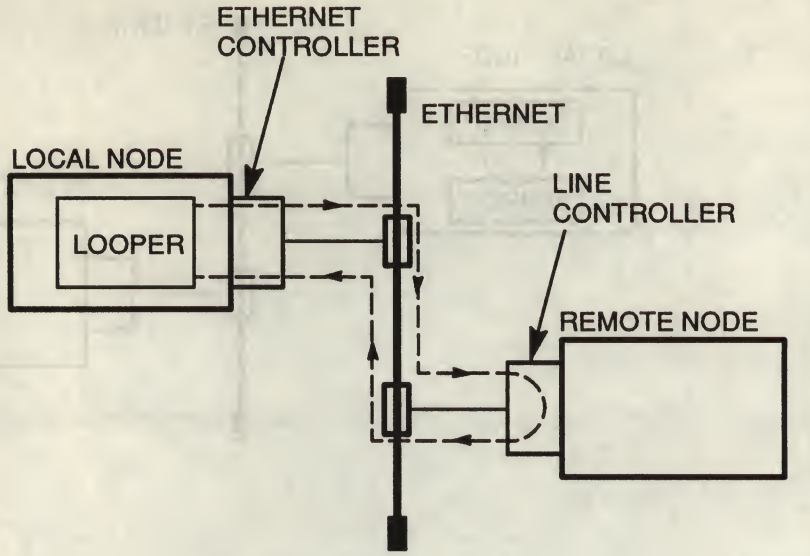


Figure 4–6: Node-Level Loopback Test to Remote Node

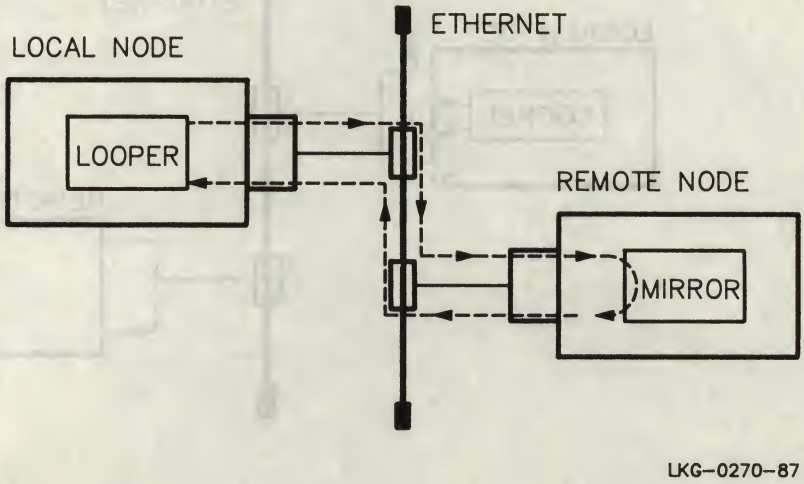


Figure 4–7: DECnet File Transfer to Local Node

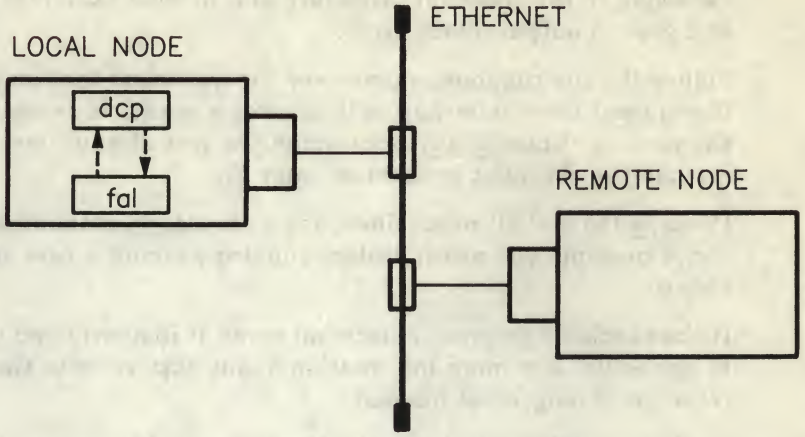
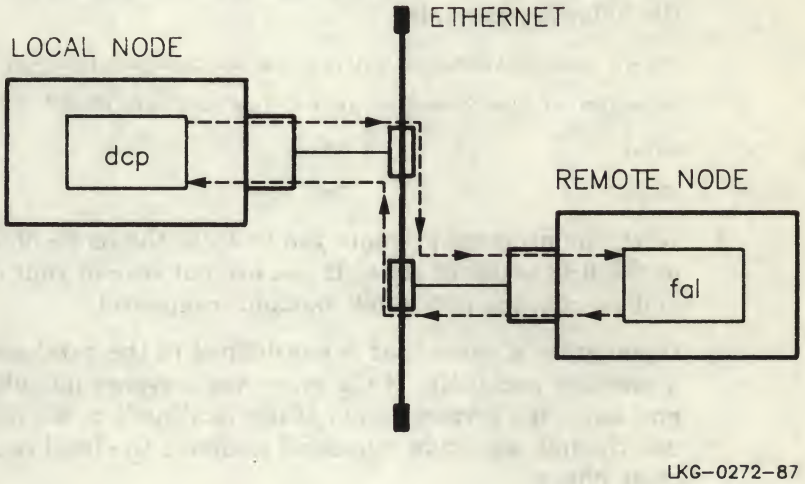


Figure 4–8: DECnet File Transfer to Remote Node



4.2 DECnet-ULTRIX Installation Checkout Procedure

To perform the checkout procedure described in this chapter, you must be a superuser and have a guest account on your system. The following pages describe each step in the checkout procedure and include examples of user input (red type) and system output (black type).

Follow the instructions and answer the questions that display on your terminal. If you need more information to answer a question, enter a question mark (?) at the prompt. Enter CTRL/Z to interrupt the test at any time. When you are ready to resume the checkout procedure, enter **fg**.

Press RET to end all input lines. If you press RET without entering any input, the script prompts you again, unless you are waiving a password specification (see step 6).

If the checkout program detects an error, it instructs you to use an **ncp** command to correct it. For more information about **ncp**, refer to the *DECnet–ULTRIX Network Management* manual.

1. Enter **/usr/etc/dnet_check**. If you do not have a guest account set up on your system, the checkout program displays a message and exits so that you can create one. Once the checkout program confirms that you have a guest account, it displays some introductory text and then checks to see that **dcp**, **drm**, and **ncp** are properly installed. If they are not, the checkout program displays a message. You must then reinstall DECnet on your ULTRIX system before continuing (see Chapter 2 for basic installation procedures and Chapter 3 if you are installing DECnet-ULTRIX into a diskless environment).

2. If **dcp** and **ncp** are properly installed, the checkout program prompts you to specify the device that you want to test (UNA-*n*, QNA-*n*, DMC-*n*, DMV-*n*, BNT-*n*, SVA-*n*, or XNA-*n*, where *n* is a value from 0 to 65535). The program then displays summary information showing the device and its state, as in the following example:

```
Enter communication device to be tested (dev-c): una-0 RET
Line Volatile Summary as of Tue Mar 26 10:57:30 EST 1990
Line                    State
UNA-0                   On
```

3. Next, the program prompts you to enter the name of your local node for use in the first series of tests. If you are not sure of your node name, enter CTRL/Z and execute the **ncp show executor** command.

   If you enter a name that is not defined in the database, the program displays a message and exits. If the error was a typing mistake, rerun **dnet_check** and enter the correct name. If the local node is not defined in the database, use the **ncp set node** command to define the local node before you rerun **dnet_check**.

   When you enter a valid local node name, the program displays summary information about the node.

```
Enter the name of your local node: boston RET
Node Volatile Summary as of Tue Mar 26 10:59:45 EST 1990
Executor node =  3.18 (BOSTON)
State = On
Identification = DECnet-ULTRIX System
```

4. The checkout program prompts you to enter the name of a remote node for use in testing. Specify a DECnet node on your Ethernet or DDCMP point-to-point wire. You can display a list of known nodes by entering CTRL/Z and executing the **ncp show known nodes** command. This command lists all nodes known to your node, which can include those beyond your local Ethernet or DDCMP point-to-point wire. You must remember which nodes are on your local wire and specify one of those for the test.

```
Enter the remote node to be used in test: CTRL/Z
Stopped
# ncp show known nodes RET

Known Node Volatile Summary as of Tue Mar 26 10:59:45 EST 1990

Executor node =  3.18 (BOSTON)

State = On
Identification = DECnet-ULTRIX System
Node    State    Active    Delay    Circuit    Next Node
Links
.
.
3.20 (SALEM)                        UNA-0      3.4 (STOW)
3.21 (WESTON)                       UNA-0      3.4 (STOW)
3.22 (GROTON)                       UNA-0      3.4 (STOW)
.
.
# fg RET
dnet_check
salem
```

If you enter a name that is not defined in the database, the checkout procedure displays a message and exits. If the error was a typing mistake, or if you want to specify a different remote node that is defined, rerun **dnet_check**. If you want to use a remote node not currently defined in the database, use the **ncp set node** command to define the node before you rerun **dnet_check**.

When you enter a valid remote node name, the system displays summary information for that node. If your Ethernet has a designated router, its name appears in the Next Node field of the display.

```
Node Volatile Summary as of Tue Mar 26 10:59:45 EST 1990

Node     State     Active     Delay     Circuit     Next Node
Links

3.20 (SALEM)                           UNA-0       3.4 (STOW)
```

5. After you have specified both the local node and a remote node, the checkout program begins a series of tests (see Figures 4-3 through 4-8). As it performs each test, the program displays the actual **ncp** commands that it is executing.

   The program gives you the option to bypass the second test, which is a controller loopback test. This test disconnects the communications device from the Ethernet or DDCMP point-to-point wire. Do not perform the test while connected to the host over the network using LAT (Local Area Transport), **rlogin**, or **dlogin**. Do not perform the test if you are a diskless client.

   If all tests are performed and all succeed, the program displays the following sequence of test reports:

```
Performing local DECnet node loopback tests.
ncp loop executor
ncp loop node boston length 1 count 10 with zero
ncp loop node boston length 2047 count 50

Warning:  answer "no" to this question if you are
          running as a diskless client, or you
          are logged in from a terminal server (LAT)
          as you will lose your file system and/or
          all currently connected LAT connections!

Do you want to perform a controller loopback test on
device una-0? (y = yes, n = no) y RET
ncp set line una-0 controller loopback
ncp loop circuit una-0 count 10 length 1486
ncp set line una-0 controller normal

Performing datalink loopback tests to node salem.
ncp loop circuit una-0 node salem length 5 count 20 with
ones
ncp loop circuit una-0 node salem length 1486 count 20

Performing node level loopback tests to node salem.
ncp loop node salem length 1 count 10 with zero
ncp loop node salem count 50
```

If a test fails, follow these suggestions for isolating the error condition:

- **Verify that the node is up on the network.** Check the display resulting from the **ncp show executor** command to determine the state of the local node. If the executor node state is **off**, issue an **ncp set executor state on** command to turn the executor on.

  To check the state of a remote node, consult the system manager for that node.

- **Verify that the circuit is turned on.** Check the display resulting from the **ncp show circuit** command to determine the circuit's state. If the circuit state is **off**, execute an **ncp set circuit** *circuit-id* **state on** command to turn the circuit on.

  If the circuit is on, check with the system manager at the remote node to see if the remote node is up on the network and to make sure that access control for the remote mirror is set properly.

- **Verify that your device is correctly connected to the Ethernet or DDCMP point-to-point wire.** Make sure the DDCMP wire is connected to the controller board, or that the Ethernet wire is connected to the transceiver or DELNI or DECOM.

- **Verify that access control for the target system's mirror object is properly set.** Check with the system manager on the target node to verify that connections can be made to the object mirror.

- **Verify that the service is properly enabled in the remote node, if applicable.**

- **Verify a problem with the remote node.** Rerun the checkout tests with a different remote node, if possible, and notify the original remote node of a possible problem.

- **Verify that the device's vector and csr switches are set to the correct addresses.** Make sure that the switch settings on the interface board of your device match the addresses defined in the system configuration file for your system. For more information, consult the user's guide for your device and refer to the ULTRIX system documentation for information on building configuration files.

- **Device failure.** Run the designated stand-alone diagnostics for the device, or call your local field service representative.
- **Rebuild the kernel.** Refer to the ULTRIX base system documentation for details.
- **Reinstall DECnet.** Follow the instructions in Chapter 2 if you are installing DECnet–ULTRIX onto a node and Chapter 3 if you are installing DECnet-ULTRIX into a diskless environment.

After you have corrected the error, rerun **/usr/etc/dnet_check**.

6. When the checkout program has successfully performed the preceding series of tests, it displays instructions for a second set of tests. It prompts you to enter the node name, user name, and password for the nodes you want to test. Enter this information for your local node first (see Figure 4-7). When you have completed that test, the program asks whether you want to test another node. Enter the corresponding data for a remote node (see Figure 4-8). You can repeat this test for as many remote nodes as you want.

   For security reasons, your response to the password prompt does not echo. However, you can still display help information about passwords by typing a question mark (?). If no password is required on the account, just press RET. If the program accepts your specification without a password, it displays a message saying that it will not use a password in the test.

   After you enter the data for the node you want to test, the program copies the file dnet_check to the specified node and back again. The checkout program then compares the contents of the original file to the one that was returned during the test. If the two files differ or if the test fails for some other reason, the program displays a message and prompts you for another node name. When the test completes, the program deletes the test file from your directory. If it cannot delete the test file, the checkout program displays a message asking you to delete the file when you complete the checkout test procedure.

```
Enter name of node for test: boston RET
Enter user login-name on node boston: jean RET
Enter password for user jean on node boston:  RET
dcp dnet_check boston/jean/password/::dnetcheck.rnd
dcp boston/jean/password/::dnetcheck.rnd dnetcheck.lnd
cmp -s dnet_check dnetcheck.lnd
drm boston/jean/password/::dnetcheck.rnd
```

7. After you have performed the test described in step 6 on your local node, you can perform it on as many remote nodes as you want. When you are finished, enter **n** in response to the prompt for another test.

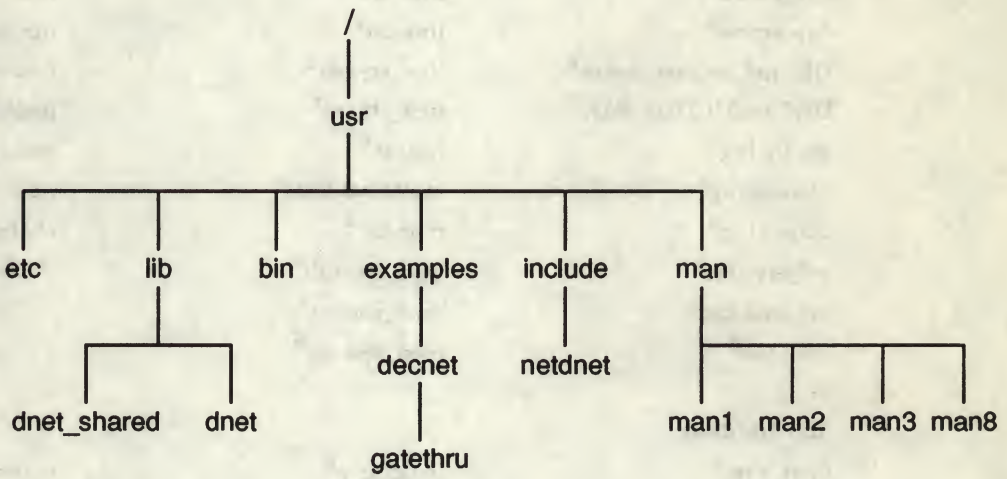
```
Do you want to run this test with another
node? (y = yes, n = no) n RET

The installation test is finished.
Your current working directory is /etc.
#
```

Appendix A

DECnet–ULTRIX Distribution Files

This appendix lists the DECnet–ULTRIX distribution files by directory. Figure A-1 shows the overall structure of the directories.

Figure A–1: DECnet–ULTRIX Directory Tree



NOTE

Digital Equipment Corporation supplies some of the files discussed in this appendix on an "as-is" basis for general customer use. Note that Digital does not offer any support for them, nor does Digital cover them under any of its support contracts.

Table A-1 shows the DECnet–ULTRIX distribution files by directory. The following key indicates which software subset each file belongs in:

B = DECnet-ULTRIX Base System
G = DECnet-Internet Gateway
U = Unsupported Utilities
M = On-Line Manual Pages

Table A–1: DECnet–ULTRIX Distribution Files by Directory

/usr/etc:		
dlogind[B]	evl[B]	mir[B]
dnet_check[B]	fal[B]	nml[B]
dnet_spawner[B]	ftpd.gw[G]	telnetd.gw[G]
dtermd[B]	GetnodeentServer	update_nodes[U]
dtr[B]	mail11dv3[B]	
/usr/lib:		
libdnet_p.a[B]	libdnet.a[B]	
/usr/lib/dnet_shared:		
area.txt[B]	exec.txt[B]	mod_xp5_ctr.txt[B]
circ.tab[B]	exec_ctr.tab[B]	mod_xs5.tab[B]
circ.txt[B]	exec_ctr.txt[B]	mod_xs5.txt[B]
circ_ctr.tab[B]	keyword_table.txt[B]	mod_xs9.tab[B]
circ_ctr.txt[B]	line.tab[B]	mod_xs9.txt[B]
dap.errors[B]	line.txt[B]	nm_help.txt[B]
DECnet_release_notes[B]	line_ctr.tab[B]	node.tab[B]
DECnet-ULTRIX.PAK[B]	line_ctr.txt[B]	node.txt[B]
dtshlp.lng[B]	log.txt[B]	node_ctr.tab[B]
dtsmsg.lng[B]	mail11v3.fatal[B]	node_ctr.txt[B]
dtsprs.lng[B]	mod.txt[B]	obj.tab[B]
evl_errs.txt[B]	mod_xac.tab[B]	obj.txt[B]
evl_evts.txt[B]	mod_xac.txt[B]	
exec.tab[B]	mod_xp5.txt[B]	
/usr/lib/dnet:		
dnet_vers[B]	logging_p[B]	objects_p[B]
exec_v[B]		
/usr/lib/lint:		
llib-ldnet.ln[U]	llib-ldnet[U]	
/usr/bin:		
dcat[B]	drm[B]	netwatch[U]
dcp[B]	dts[B]	nfi[U]
dlogin[B]	mail11v3[B]	nodename[B]
dls[B]	ncp[B]	tell[U]

(continued on next page)

Table A–1 (Cont.): DECnet–ULTRIX Distribution Files by Directory

/usr/examples/decnet:

dnet_echo1.c[U]
dnet_echo1d.c[U]
dnet_echo2.c[U]
dnet_echo2d.c[U]
tell.c[U]
TELL.COM[U]

/usr/examples/decnet/gatethru:

gatewayd[U]
gatewayd.c[U]
Makefile[U]
README[U]

/usr/examples/dli:

dli_802.c[U]
dli_802d.c[U]
dli_eth.c[U]
dli_ethd.c[U]
dli_setsockopt.c[U]

/usr/man/man1:

dcat.1dn[M]
dcp.1dn[M]
dlogin.1dn[M]
dls.1dn[M]
drm.1dn[M]
errors.1dn[M]
nodename.1dn[M]
nfi.1dn[M]
tell.1dn[M]

/usr/man/man2:

accept.2dn[M]
bind.2dn[M]
close.2dn[M]
connect.2dn[M]
getpeername.2dn[M]
getsockname.2dn[M]
getsockopt.2dn[M]
listen.2dn[M]
read.2dn[M]
recv.2dn[M]
select.2dn[M]
send.2dn[M]
setsockopt.2dn[M]
shutdown.2dn[M]
socket.2dn[M]
write.2dn[M]

/usr/man/man3:

dnet_addr.3dn[M]
dnet_conn.3dn[M]
dnet_eof.3dn[M]
dnet_getalias.3dn[M]
dnet_htoa.3dn[M]
dnet_ntoa.3dn[M]
dnet_otoa.3dn[M]
getnodeadd.3dn[M]
getnodeent.3dn[M]
getnodename.3dn[M]
nerror.3dn[M]

/usr/man/man8:

dts.8dn[M]
ncp.8dn[M]

/usr/include/netdnet:

dn.h[B]
dnetdb.h[B]
node_params.h[B]
x25db.h[B]

Note: If you are installing DECnet–ULTRIX onto a RISC processor, you do not receive the **/usr/lib/libdnet_p.a** file.

Index

HOW TO ORDER ADDITIONAL DOCUMENTATION

DIRECT TELEPHONE ORDERS

In Continental USA
call 800–DIGITAL

In Canada
call 800–267–6215

In New Hampshire
Alaska or Hawaii
call 603–884–6660

In Puerto Rico
call 809–754–7575

ELECTRONIC ORDERS (U.S. ONLY)

Dial 800–DEC–DEMO with any VT100 or VT200
compatible terminal and a 1200 baud modem.
If you need assistance, call 1–800–DIGITAL.

DIRECT MAIL ORDERS (U.S. and Puerto Rico*)

DIGITAL EQUIPMENT CORPORATION
P.O. Box CS2008
Nashua, New Hampshire 03061

DIRECT MAIL ORDERS (Canada)

DIGITAL EQUIPMENT OF CANADA LTD.
940 Belfast Road
Ottawa, Ontario, Canada K1G 4C2
Attn: A&SG Business Manager

INTERNATIONAL

DIGITAL
EQUIPMENT CORPORATION
A&SG Business Manager
c/o Digital's local subsidiary
or approved distributor

Internal orders should be placed through the Software Distribution Center (SDC),
Digital Equipment Corporation, Westminster, Massachusetts 01473

*Any prepaid order from Puerto Rico must be placed
with the Local Digital Subsidiary:
809–754–7575 x2012

HOW TO ORDER ADDITIONAL DOCUMENTATION

DIRECT TELEPHONE ORDERS

ELECTRONIC ORDERS (U.S. ONLY)

DIRECT MAIL ORDERS (U.S. and Puerto Rico)

DIRECT MAIL ORDERS (Outside)

INTERNATIONAL

DECnet–ULTRIX
Installation
AA–EA87E–TE

READER'S COMMENTS

What do you think of this manual? Your comments and suggestions will help us to improve the quality and usefulness of our publications.

Please rate this manual:

	Poor				Excellent
Accuracy	1	2	3	4	5
Readability	1	2	3	4	5
Examples	1	2	3	4	5
Organization	1	2	3	4	5
Completeness	1	2	3	4	5

Did you find errors in this manual? If so, please specify the error(s) and page number(s).

_____

_____

_____

_____

General comments:

_____

_____

_____

_____

Suggestions for improvement:

_____

_____

_____

_____

Name _____ Date _____

Title _____ Department _____

Company _____ Street _____

City _____ State/Country _____ Zip _____

DO NOT CUT – FOLD HERE AND TAPE

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY LABEL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

digital™

Networks and
Communications Publications
550 King Street
Littleton, MA 01460–1289

DO NOT CUT – FOLD HERE

CUT ON THIS LINE

digital

DECnet–ULTRIX

Use

May 1990

This manual shows how you use the DECnet–ULTRIX user commands to perform file transfer and other user tasks.

Supersession/Update Information:	This is a new manual.
Operating System and Version:	ULTRIX V4.0
Software Version:	DECnet–ULTRIX V4.0

digital™

Order Number: AA–PB61A–TE

AA–PB61A–TE
May 1990

This manual was produced by Networks and Communications Publications.

Contents

Preface

This manual contains both tutorial and reference material for DECnet–ULTRIX end users.

Intended Audience

This manual is for anyone who wants to use DECnet–ULTRIX to log on to remote DECnet nodes, exchange mail with users on remote DECnet nodes, and work with files on DECnet nodes. You should be familiar with the ULTRIX operating system and general file transfer principles. You should also know how to work on the DECnet nodes you plan to access via remote login.

Structure of This Manual

This manual contains six chapters and two appendixes:

- Chapter 1 introduces DECnet–ULTRIX product features and utilities.
- Chapter 2 describes how to use the **dlogin** command to log on to other DECnet systems.
- Chapter 3 describes how to use the ULTRIX **mail** utility to exchange mail with DECnet nodes.
- Chapter 4 explains how to specify remote files on a DECnet node.
- Chapter 5 describes how to display, copy, and delete files and directories on a remote DECnet node.
- Chapter 6 describes the DECnet–ULTRIX commands in detail. This reference chapter is also available on line using the **man** command.
- Appendix A lists the DECnet–ULTRIX error messages.
- Appendix B shows sample DECnet file specification formats.

Related Documents

To supplement the *DECnet–ULTRIX Use* manual, see the following DECnet–ULTRIX documents:

- *DECnet–ULTRIX Release Notes*

  This document contains miscellaneous information and updates not included in other books in the DECnet–ULTRIX documentation set.

- *DECnet–ULTRIX Installation*

  This manual describes procedures for installing, customizing, and testing a DECnet–ULTRIX node for proper operation. This manual also lists the DECnet–ULTRIX distribution files and the directory path names.

- *DECnet–ULTRIX DECnet–Internet Gateway Use and Management*

  This manual describes how to install, use, and manage the DECnet–Internet Gateway.

- *DECnet–ULTRIX Programming*

  This manual explains concepts and offers guidelines for application programming in the DECnet–ULTRIX programming environment. This manual also describes DECnet–ULTRIX system calls and subroutines, and shows DECnet–ULTRIX data structures and programming examples.

- *DECnet–ULTRIX Network Management*

  This manual describes procedures for managing the network, such as defining permanent and volatile databases, defining node identifications and addresses, defining lines and circuits, enabling event logging, displaying network counter information, operating and controlling a DECnet–ULTRIX node, and testing the network operation.

- *DECnet–ULTRIX NCP Command Reference*

  This manual describes the Network Control Program (**ncp**) commands for defining, monitoring, and testing your network.

To obtain a detailed description of the Digital Network Architecture, see *DECnet Digital Network Architecture (Phase IV), General Description.*

For a beginner's introduction to the ULTRIX operating system, see *The Little Gray Book: An ULTRIX Primer.*

Graphic Conventions

This manual uses the following conventions:

Convention	Meaning
special	In running text, ULTRIX commands, command options, user names, file names, and directory names appear in **special** type.
`example`	Indicates an example of system output or user input. System output is in black type; user input is in red type.
lowercase/UPPERCASE	Because the ULTRIX software is case sensitive, you must type all literal input in the case shown. In running text, UPPERCASE is also used for the names of all DECnet nodes, including DECnet–ULTRIX nodes. This convention follows DECnet protocol, which names and recognizes all nodes in UPPERCASE. However, node names are not case sensitive and need not be typed in the case shown.
italic	Indicates a variable, for which either you or the system specifies a value.
[ ]	On command syntax lines, square brackets indicate optional arguments. Do not type the brackets.
key	Indicates a key on your keyboard. CTRL/*key* represents a CONTROL key sequence, where you press the CONTROL key at the same time as the specified key. Note that keyboard keys are represented by this symbol, <*key*>, on line.
RET	Indicates the RETURN key.
%	The percent sign, the standard ULTRIX system prompt, is used in all examples in this manual to indicate an ULTRIX system.
$	The dollar sign, the standard VMS system prompt, is used in all examples in this manual to indicate a VMS system.

NOTE

All numbers are decimal unless otherwise noted.

Terminology

In this manual, "DECnet–RSX" stands for any of these DECnet products: DECnet-11M-PLUS, DECnet–Micro/RSX, DECnet-11S, DECnet-11M.

The following acronyms are used in this manual:

DDCMP	Digital Data Communications Message Protocol
DNA	Digital Network Architecture
FTP	File Transfer Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol

Chapter 1

Introduction to the DECnet–ULTRIX Product

The DECnet–ULTRIX product is software that runs on an ULTRIX system. With DECnet–ULTRIX software, an ULTRIX system can act as an end node on a DECnet network. The DECnet–ULTRIX product is an end-node implementation of Digital Network Architecture (DNA) Phase IV.

You can connect your DECnet-ULTRIX system to a DECnet network either through Digital Data Communications Message Protocol (DDCMP) point-to-point line or a DECnet Ethernet cable. A DECnet–ULTRIX node can access all other Phase III and Phase IV **Decnet nodes** on its network through direct communication with a Phase IV routing node. The routing node for a DECnet–ULTRIX system is either the adjacent node on its DDCMP point-to-point line or one of the nodes on the same Ethernet.

DECnet-ULTRIX software offers numerous features and capabilities:

- User commands that support standard ULTRIX conventions, including support of regular expressions, wildcards, and metacharacters. DECnet–ULTRIX software offers remote login to remote DECnet nodes; mail exchange; and file access and transfer between systems. The DECnet–ULTRIX user commands and utilities are documented in this manual.
- Software that supports the DECnet–Internet Gateway, which offers **bidirectional gateway** functions between DECnet and *Internet* systems, including file transfer, remote log-in, and mail.

  For example, an VMS user on a DECnet node and a UNIX user on an Internet node can display, copy, and delete each other's files; exchange mail; and log in to each other's systems. (The Gateway offers VMS users access to the powerful resources of the ULTRIX environment.) For more information about the DECnet–Internet Gateway, see the *DECnet–ULTRIX DECnet–Internet Gateway Use* manual.
- A programming interface that lets you write cooperating programs to exchange data over a DECnet network. For more information about DECnet–ULTRIX programming, see the *DECnet–ULTRIX Programming* manual.
- Network management features that let you configure a DECnet–ULTRIX node, test network performance, monitor network activity, and manage network components. For more information about network management, see the *DECnet–ULTRIX Network Management* manual and the *DECnet–ULTRIX NCP Command Reference*.

The following sections describe these features and capabilities.

1.1 DECnet–ULTRIX User Commands and Utilities

Table 1-1 shows the tasks DECnet–ULTRIX users can perform and describes the commands associated with those tasks.

Table 1–1: Overview of DECnet–ULTRIX User Commands and Utilities

Task	Command	Description
Remote login	**dlogin**	The **dlogin** command lets you log on to remote DECnet nodes so that you can access the resources of those systems. DECnet–ULTRIX software also lets remote DECnet users log on to your node. For more information, see Chapter 2 or 6.
Mail exchange	**mail**	The ULTRIX **mail** utility lets you exchange messages with other DECnet users. For more information, see Chapter 3 and *The Little Gray Book: An ULTRIX Primer*.
Directory listing	**dls**	The **dls** command lets you examine the contents of remote directories. DECnet–ULTRIX software also lets remote DECnet users display the contents of your directories. For more information, see Chapter 5 or 6.
File display	**dcat**	The **dcat** command displays the contents of one or more remote DECnet files. DECnet–ULTRIX software also lets remote DECnet users display the files on your system. For more information, see Chapter 5 or 6.
File transfer	**dcp**	The **dcp** command lets you copy files to and from remote DECnet systems. Remote DECnet users can initiate copy requests to your node. For more information, see Chapter 5 or 6.
File deletion	**drm**	The **drm** command lets you delete files from remote directories. Remote DECnet users can delete files from your local directories. For more information, see Chapter 5 or 6.

1.2 DECnet–Internet Gateway Features

The DECnet–Internet Gateway offers the following features:

Telnet

You can use Telnet commands through the Gateway. Telnet is the standard Internet application protocol for remote login. Using Telnet through the Gateway, Internet users can log on to remote DECnet nodes.

File Transfer Protocol

You can use File Transfer Protocol (**ftp**) through the Gateway. **ftp** is the primary Internet standard for file transfer. Using **ftp** through the Gateway, Internet users can access and manipulate files and directories on remote DECnet nodes.

Mail systems	You can use VMS, ULTRIX, and UNIX mail systems through the Gateway. Specify recipients according to the addressing syntax described in the *DECnet–ULTRIX DECnet–Internet Gateway Use* manual.
Common DECnet commands	You can use various common DECnet commands through the Gateway. See Chapter 3 for sample DECnet-VAX commands.

1.3 DECnet–ULTRIX Programming Interface

The DECnet–ULTRIX programming interface offers the following features:

Client-server communication	Sometimes called task-to-task communication, client-server communication lets DECnet–ULTRIX applications communicate with remote Phase III and Phase IV DECnet applications through a socket-level programming interface.
DECnet and TCP/IP coexistence	DECnet protocols and Transmission Control Protocol/Internet Protocol (TCP/IP) coexist and share system resources, including Ethernet and DDCMP hardware. You can easily modify most TCP/IP programs to use DECnet protocols, or DECnet programs to use TCP/IP protocols. You can use DECnet and TCP/IP simultaneously on an Ethernet, and you can alternate between the two protocols on DDCMP point-to-point lines.
File access	Programs on any other DECnet Phase III or Phase IV system can access DECnet–ULTRIX files and directories for sequential reading, writing, or deletion.

1.4 DECnet–ULTRIX Network Management Features

The DECnet–ULTRIX network management software offers the following features:

Node configuration	You can configure your DECnet–ULTRIX node to ensure that it runs smoothly on the network.
Network performance testing	You can test your DECnet–ULTRIX node's performance on the network.
Network Control Program (ncp)	You can manage the components of your network and test network performance. You can also display information on the condition, characteristics, and performance of network components.

1.5 Similarities Between DECnet–ULTRIX and ULTRIX

Both new and experienced ULTRIX users can easily become familiar with the DECnet–ULTRIX user environment because:

- DECnet–ULTRIX commands support standard ULTRIX conventions, including the use of regular expressions, wildcards, and metacharacters.

- You can **pipe** or **redirect** the DECnet–ULTRIX commands' input or output according to standard ULTRIX conventions.
- The DECnet–ULTRIX mail system utilizes the ULTRIX **mail** interface, requiring ULTRIX users to learn only a new syntax for creating mail recipients' addresses.
- DECnet–ULTRIX provides on-line manual pages (also called *man pages*) that follow standard ULTRIX man page conventions. The DECnet–ULTRIX manual pages, however, use a "dn" page identifier to distinguish them from ULTRIX manual pages. For example:

  To see the ULTRIX base system manual page for the **write** command, type:

  ```
  % man write RET
  ```

  To see the ULTRIX base system manual page for the **write** system call, type:

  ```
  % man 2 write RET
  ```

  To see the DECnet–ULTRIX manual page for the **write** system call, type:

  ```
  % man 2dn write RET
  ```

  The following section provides more information about on-line manual pages.

1.6 Displaying On-Line Manual Pages

On-line manual pages are pages from reference manuals that you can display on your screen. See the on-line manual pages for detailed descriptions of all the DECnet–ULTRIX commands. (Note that the DECnet–ULTRIX manual pages also appear in the DECnet–ULTRIX Command Summary (Chapter 6) of this manual.

To display an on-line manual page for a specific command, enter **man** and the command name. You can even display a reference page describing the **man** command itself. Enter:

```
% man man RET
```

The system displays:

```
                                                              man(1)

NAME
     man - displays manual pages on-line

SYNTAX
     man -k keyword...
     man -f page_title...
     man [-] [-t] [-s] [1...8] page_title...

DESCRIPTION
     The man command is a program that provides on-line displays
     of the reference pages.  Using options, you can direct the
     man command to display one line summaries of reference pages
     which contain a specific keyword, or you can use this com-
     mand to display one line summaries of specific reference
     pages.

OPTIONS
     -k                  display one line summaries of each
                         reference page that contains the speci-
                         fied keyword or keywords.

--More--(27%)
```

The **–More–** symbol indicates that there is more information available. Press the space bar or RET to see more, or type **q** (without pressing RET) to quit. The **(27%)** symbol indicates that 27% of the information available on the **man** command is displayed. The **man(1)** symbol at the top of the display is the page title.

NOTE

To use the **man** command, you must have the ULTRIX manual pages installed on your local system.

Chapter 2

Logging On to a Remote DECnet Node

Using the **dlogin** command, you can log on to a remote DECnet node and use programs running on that system. The remote DECnet node can be another DECnet–ULTRIX system or a different operating system running DECnet, such as a VMS system. (In this chapter, your target node is referred to as a **remote node**. Your target node, however, could just as easily be your **local node**.)

This chapter describes how you can use **dlogin** to establish and control a login session with a remote DECnet node. This chapter tells you how to:

- Log on to a remote DECnet node
- Log off a remote DECnet node
- Enter local command mode
- Display a list of **dlogin** commands
- Suspend the **dlogin** session
- Record, or log, the **dlogin** session
- Select a new escape character
- Use the escape character without entering local command mode

Be sure you understand how to use **dlogin** before you log on to a remote DECnet node. See the next section.

2.1 Understanding dlogin

This section describes the **dlogin** session you create when you log on to a remote DECnet node.

Your local node is your local DECnet–ULTRIX system. When you log on to this node, you start a **local login session**. During this local login session, you can use the **dlogin** command to log on to a remote DECnet node. A remote DECnet node is any node in the network that is running DECnet software. When you log on to a remote node, you start a remote login session. That session is called the **dlogin** session because you use the DECnet login command, **dlogin**, to initiate it.

During the **dlogin** session, you can run programs and access resources available on the remote system, according to whatever privileges the remote system grants you. You can also interrupt your **dlogin** session and execute commands on your local node. To do this, you use the **dlogin** escape character. The escape character can be any keyboard character; the default escape character is the **tilde** (~).

When you type the escape character and press RET, the escape character is not echoed to your screen or even read by the remote node. Instead, the local node, which controls your **dlogin** session, reads the escape character and interprets it as your signal to interrupt your activities on the remote node.

When you interrupt your **dlogin** session, you enter local command mode. Local command mode is simply an interface to your local node. The **dlogin** session is still active while you are in local command mode. Local command mode is characterized on screen by the **local command>** prompt.

At the **local command>** prompt, you can issue any ULTRIX or DECnet–ULTRIX command you want to execute on your local node. You can also issue the commands described in this chapter that control your **dlogin** session (for example, the commands that suspend your **dlogin** session or open a new **log file**).

The following sections show you how to begin, end, and control your **dlogin** session.

2.2 Logging On to a Remote DECnet Node

To log on to a remote DECnet node from your DECnet–ULTRIX system, type the **dlogin** command followed by the remote DECnet node name. The remote node prompts you for the login information it requires. Once you are logged on to the node, you can issue commands to that system. In the following example, user Irene logs on to the remote VMS node BACON and issues the VMS SHOW USERS command:

```
% dlogin bacon RET
Username:  IRENE RET
Password:  GOODNIGHT RET  (not echoed)
               Welcome to VAX/VMS V5.2 on node BACON
$ SHOW USERS RET
```

2.3 Logging Off a Remote Node

To log off the remote DECnet node, simply type the remote node's logout command. Logout commands vary; for example, the command is **logout** on DECnet–VAX and **bye** on DECnet–RSX. If you do not know the logout command for the remote operating system you are using, see that system's documentation.

In the following example, user Irene is shown logging off a DECnet–VAX node:

```
$ logout RET
IRENE   logged out at   6-JAN-1990   09:46:29.97
dlogin -- session terminated
%
```

Remember, when you log off the remote node, you end the **dlogin** session.

To end a **dlogin** session before you have logged on to the remote node, type the escape character (the tilde (~) is the default) and press RET. At the **local command>** prompt, press CTRL-D or type **exit** and press RET.

In this example, user Frank ends the **dlogin** session because he cannot remember the user name and **password** that have been assigned to him on the remote node BACON:

```
% dlogin bacon RET

Username: FRANK RET
Password: ORANGES RET  (not echoed)
User authorization failure
Username: ~ RET  (not echoed)

? for HELP in local command mode
local command> exit RET

dlogin -- session terminated
%
```

2.4 Entering Local Command Mode

Using the **dlogin** escape character, you can interrupt your login session on the remote node and enter local command mode.

At the **local command>** prompt, you can execute any ULTRIX or DECnet–ULTRIX command on your local node. You can also perform the following tasks, as described in this chapter:

- Display a list of **dlogin** commands
- Suspend the **dlogin** session
- Record the **dlogin** session

To enter the local command mode, type the escape character followed by RET at the remote system's prompt. (The tilde (~) is the default escape character.) The screen displays the **local command>** prompt. For example:

```
$ ~ RET  (not echoed)

? for HELP in local command mode
local command>
```

To return control to your **dlogin** session on the remote node, press RET at the **local command>** prompt, as follows:

```
local command> RET
$
```

You may have to press RET twice for the system prompt to appear.

2.5 Displaying a List of dlogin Commands

To display helpful information about the commands that control your **dlogin** session, type a question mark (?) at the **local command>** prompt. For example:

```
local command> ?
local command> ?              Displays this menu.
local command> >filename      Logs session to specified file.
local command> >>filename     Logs session, appending it to specified file.

local command> >              Closes the open log file.
local command> suspend        Suspends dlogin session.
local command> exit           Exits dlogin session.

local command> <CTRL-D>       Exits dlogin session.
local command> cmd            Invokes shell to execute a command.
local command>                (Blank line) - Resumes dlogin session.

[No log file active]
```

NOTE

In this display, *cmd* represents any ULTRIX command you can enter at the **local command>** prompt.

2.6 Suspending the dlogin Session

If you want to interrupt your **dlogin** session and return control (temporarily) to your local node, you can **suspend** your **dlogin** session. A suspended **dlogin** session is not active. As a result, the screen displays your local node's prompt, and you can execute commands at your local node.

To suspend a session, type the escape character followed by RET at the remote system's prompt. When the **local command>** prompt appears, type the **suspend** command followed by RET. For example:

```
$ ~ RET  (not echoed)

? for HELP in local command mode
local command> suspend RET

Stopped
%
```

At your local system's prompt, you can execute any commands or procedures that you usually execute on your local system.

When you are ready to resume the **dlogin** session, you simply issue a **fg** command to bring the session to the foreground and press RET at the **local command>** prompt. The following example shows a suspended session on node BACON being resumed:

```
% fg RET
dlogin bacon
local command> RET
$
```

2.7 Recording the dlogin Session

If you want to maintain a record of a **dlogin** session, you can open a log file. The log file contains both your input and the system's responses. The **dlogin** command allows you to have the session logged to a new log file or appended to an existing log file. The **dlogin** command also allows you to close the log file during the session.

2.7.1 Creating a Log File

You can open a new log file using either of the following methods:

- **When logging on to the remote DECnet node:** use the **-l** option on the **dlogin** command line and specify the name of the log file to which you want the session logged. For example, the following command line begins a **dlogin** session on node BACON and opens the file **logfile3**:

  ```
  % dlogin bacon -l logfile3 RET
  ```

- **During a session on the remote node:** enter local command mode with the escape character and, at the **local command>** prompt, issue the **>***filename* command, where *filename* specifies the name of a log file. For example, the following command opens the file **logfile3**:

```
$ ~ RET   (not echoed)

? for HELP in local command mode
local command> >logfile3 RET
local command> RET
```

If you specify an existing log file when you use either of the two preceding commands, the system replaces the contents of that file with the new log file information.

If you begin your session using the **-l** option to open a log file and later, during the session, you open a second log file with the **>***filename* command, the first log file is automatically closed.

2.7.2 Adding New Log Information to a Log File

You can add, or append, new log information to an existing log file. Enter the local command mode and issue the **>>***filename* command, where *filename* specifies the name of the existing log file. For example, the following command appends a record of the current session to the file **logfile_1990**:

```
$ ~ RET   (not echoed)

? for HELP in local command mode
local command> >>logfile_1990 RET
local command> RET
```

2.7.3 Closing a Log File

Log files are closed automatically when you log off the remote node and end the **dlogin** session. To close a log file during a **dlogin** session, enter the local command mode and issue the **>** command. For example:

```
$ ~ RET   (not echoed)

? for HELP in local command mode
local command> > RET
local command> RET
```

2.8 Selecting a New dlogin Escape character

When you start a **dlogin** session, you can select a new escape character for that session. The character you select is the escape character only for the length of that **dlogin** session. To select a different escape character, use the **-e** option on the **dlogin** command line. For example, the following command starts a **dlogin** session and sets the escape character to the circumflex (^) character:

```
% dlogin bacon -e^ RET
```

2.9 Using the Escape Character Without Entering Local Command Mode

You might want to use the escape character as part of a mail message or a command line without signaling the **dlogin** session to enter local command mode.

Type the escape character once, followed by any character (even the escape character) except RET. The escape character appears on your screen only after you have typed another character. For example:

```
%dcp -r kimono::~uucp . RET
```

or

```
% cd ~~ RET  (one ~ is echoed)
```

Chapter 3

Sending Mail to Users on Remote DECnet Nodes

You can use the ULTRIX mail command to send messages to users on remote DECnet nodes. You use the recipient's DECnet mail address as the destination for the message. This chapter defines the DECnet mail address and shows sample mail messages.

For information on the conventions used by ULTRIX **mail**, see *The Little Gray Book: An ULTRIX Primer*.

3.1 The DECnet Mail Address

The **DECnet mail address** identifies the user who will receive the mail you are sending. The DECnet mail address consists of the user's DECnet node name, a double colon (::), and the user's DECnet **user name**. For example, the DECnet mail address for user **jim** on node DAVIS is

```
davis::jim
```

You can also follow the popular Internet mail addressing syntax to identify mail recipients. Internet syntax is as follows:

```
username@node.dnet
```

where

username	is the name of the mail recipient.
node	is the name of the recipient's node.
.dnet	is the Internet pseudodomain name.

For example, the **Internet mail address** for user **Markie** on node TWICE is

```
markie@twice.dnet
```

The Internet addressing syntax is provided merely as a convenient interface for Internet users. DECnet–ULTRIX actually converts this Internet address to the DECnet syntax before sending the mail out.

3.2 Sending Mail to a Remote DECnet User

To send mail to a user on a remote DECnet node, type the DECnet–ULTRIX **mail** command and the user's DECnet (or Internet format) mail address. For example:

```
% mail davis::jim RET
```

3.2.1 Mailing a New Message

In this example, user **Helen** on a DECnet–ULTRIX node sends mail to user **Jim** on node DAVIS.

```
% mail davis::jim RET
Subject:New Position RET
RET
Dear Jim, RET
RET
You've been in Davis, California, for three years now. RET
If you are open to moving to Atlanta, we would like to RET
consider you for a position here.  Please let me know RET
by next week how you feel about this. RET
RET
                    Sincerely, RET
RET
                    Helen RET
CTRL/D
Cc:helen RET
%
```

3.2.2 Mailing a File to a Remote DECnet User

In this example, user **Jim** on node DAVIS sends the existing file **resume** to user **Helen** on remote DECnet–ULTRIX node ATLANT. **Jim** does not send a copy of the message to anyone.

```
% mail atlant::helen RET
Subject: helen, here is my updated resume RET
~r resume RET
"resume" 23/1204. RET
CTRL/D
Cc: RET
%
```

3.2.3 Special Considerations

Be aware that a VMS node will not accept a message from ULTRIX if the "TO:" field of the message exceeds 512 characters. To reduce the number of characters in this field, use a **sendmail** alias.

For more details about setting up an **alias**, see the description in **/usr/lib/aliases**.

Chapter 4

Specifying Files on Remote DECnet Nodes

Before you can view or work with remote files and directories, you must be able to specify those files and directories. This chapter explains how to specify remote files and directories using the following information:

- File specification formats for all the operating systems that run DECnet
- Wildcard characters for use in file specifications
- **Access-control information** required to access files on a remote node
- Short cuts for specifying the full access-control string in the file specification

The examples in this chapter illustrate three DECnet–ULTRIX commands, **dcat**, **dcp**, and **dls**. In these examples, **dcat** displays the contents of files, **dcp** copies files, and **dls** lists the contents of directories. For more information about these commands, see Chapters 5 and 6.

4.1 File Specification Formats

Using DECnet–ULTRIX commands, you can work with files that reside on remote ULTRIX and **non-ULTRIX DECnet nodes**. The format you use to specify the files will vary according to the type of system the file resides on.

Table 4–1 lists the file specification formats for files on ULTRIX and non-ULTRIX DECnet nodes. For more information on these file specifications, refer to Appendix B.

Table 4–1: File Specifications for DECnet Nodes

DECnet System	File Specification
DECnet–VAX	*node::'dev:[directory]filename.typ;ver'*
DECnet–RSX	*node::'dev:[ufd]filename.typ;ver'*
DECnet–IAS	*node::'dev:[ufd]filename.typ;ver'*
DECnet/E	*node::'dev:[ppn]filename.typ'*
DECnet-10	*node::'dev:[ufd]filename.ext[p,pn]<prot'*
DECnet-20	*node::'dev:<directory>filename.typ.gen;att'*
DECnet–RT	*node::'dev:filename.typ'*
DECnet–DOS	*node::'dev:\path\filename.typ'*
DECnet–ULTRIX	*node::'path/filename'*

In Table 4–1, all the information following the node name is enclosed in single quotation marks (' '). The quotation marks prevent the local shell (or command interpreter) from reading and interpreting certain special characters (or metacharacters) which are included in the specification. Only the remote system reads and interprets characters enclosed in quotes.

Metacharacters represent special commands or functions to the local shell. Here are the ULTRIX metacharacters:

Table 4–2: ULTRIX Metacharacters

Character	Meaning
< >	Angle brackets
&	Ampersand
*	Asterisk
\	Backslash
|	Bar
{ }	Braces
	Blank space
$	Dollar sign
!	Exclamation point
( )	Parentheses
?	Question mark
;	Semicolon
[ ]	Square brackets

You must do something to ensure that the metacharacters you include in file specifications are not interpreted by the local shell. You can (as shown in the table) choose to enclose all the information following the node name in single quotes as a standard part of your file specification. The advantage of this choice is that you do not have to remember which characters require special treatment. Of course, you can also choose to treat the special characters individually by enclosing them in single quotes or preceding each one with a backslash.

The following example shows several ways you can compose a command line that includes metacharacters (square brackets and an asterisk):

```
% dcat woods::'usrdsk2:[nature]trees.txt;*'
```

or

```
% dcat woods::usrdsk2:\[nature\]trees.txt;\*
```

or

```
% dcat woods::usrdsk2:'[nature]'trees.txt;'*'
```

4.2 Using Wildcard Characters

All Digital operating systems support the use of wildcard characters. You can use them in file specifications to refer to a group of files by a general name, rather than specifying each file individually. Table 4-3 describes the ULTRIX-specific wildcard characters.

Table 4–3: Wildcard Characters

Character	Meaning
*	Matches one or more characters, except a slash (/).
?	Matches a single character, except a slash (/).
[set]	Matches one character from a set. Set members can be enumerated (for example, [1234]) and/or contain ranges (for example, [1-4]) where the ASCII order is used.
{tokens}	Matches a string from a list of ASCII strings, separated by commas, which can be a portion of the filename. For example, {abc,def}* will match any **file name** that begins with abc or def. A token can contain other wildcard characters.

These wildcard characters are among the metacharacters listed in Table 4-2. If you include a wildcard in a file specification, enclose it in single quotes or precede the wildcard with a backslash (\). Otherwise, the local shell will read and interpret the wildcard character.

This example shows two ways you can compose a command line that includes the asterisk (*) wildcard character:

```
% dcat fact::'dev233:*.txt'  RET
```

or

```
% dcat fact::dev233:\*.txt  RET
```

Table 4–3 describes how the ULTRIX system interprets wildcard characters. See the documentation supplied with the remote system for information on how that system interprets wildcard characters.

4.3 Supplying Access-Control Information

DECnet nodes can use access-control information to screen connection requests from remote nodes. A node screens connection requests by checking this user-supplied information against information in local password or proxy files.

Access-control information consists of a **user name**, **password** (optional), and **account** (optional). You can specify this information in the following ways:

- Enter the access-control information manually.
- Use an alias.
- Use a proxy account.

The target system handles access-control information as follows:

- When you supply access-control information to a remote system, the remote system checks its password file to verify the user name and password you gave it.
- When you supply only a user name, the remote system checks its proxy file. If the user name is not defined in the proxy file, the remote system checks if the user name belongs to an account that has no password.
- When you omit all access-control information, the remote system checks to see if you are defined in its proxy file. If not, the system then tries to use its default access account.

NOTE

The ULTRIX system is case sensitive. Non-ULTRIX systems may pass access-control information in uppercase. Therefore, if you plan to pass access-control information from a non-ULTRIX to an ULTRIX system, consult the other system's documentation to see whether or not the system provides a **case-sensitive** means for passing this information. If it does not, add an uppercase account name and password to your ULTRIX system password file.

The following systems pass access-control information in uppercase:

- DECnet–IAS
- DECnet–RT
- DECnet-11M V4.0 or earlier
- DECnet-11M-PLUS V2.0 or earlier

4.3.1 Entering the Information Manually

You can manually specify access-control information following the node name in the file specification. Use this format:

node/username/password/account::file_information

where

node	is the name or address of the DECnet node.
username	is a string of up to 39 characters identifying the user's log-in name, which is verified by the remote node.
password	is a string of up to 39 characters needed for gaining access to the remote system.
account	is a string of up to 39 characters that is verified by the remote node's system account file. (The *account* field is ignored by most DECnet systems.)
file_information	is the rest of the file specification as specified in Table 4–1.

4.3.1.1 User Name and Password

When you include the user name and password in the file specification, the remote node can verify access clearance and then execute the command. This example does not include file information because ATLANT/jones indicates the login directory to be listed by the **dls** command:

```
% dls atlant/jones/mysecret:: RET

Directory ATLANT::/usr/users/jones/
MYFILE          bin             printf.notes        filen
log             log2
%
```

4.3.1.2 User Name Only (Password Required)

If you include only the user name in the file specification, DECnet–ULTRIX prompts you for the password. The advantage of letting the system prompt you for the password is that your password is not displayed on your screen as you type it, thus providing additional security.

In the following example, a file in account **don** on node RED is copied to account **sue** on node BLUE. The system prompts for passwords to both accounts:

```
% dcp red/don::'dsk3:[don]num.dat' blue/sue::'u$2:[sue.data]02.dat' RET
Password for red/don:: ? subway RET  (not echoed)
Password for blue/sue:: ? underground RET  (not echoed)
%
```

4.3.1.3 User Name Only (Password Not Required)

A password is not required along with the user name if the account does not have a password or if a proxy account exists. However, to avoid being prompted for a password, type a slash (/) immediately after the user name. This indicates that you are not specifying a password, so the system does not prompt you for one. For example:

```
% dls atlant/public/:: RET

Directory ATLANT::/usr2/users/public/
README              bin                 printf.notes        filen
logfile1            logfile2
%
```

If you omit the slash from the command line and the system prompts you for a password, simply press RET at the password prompt. For example:

```
% dls atlant/public:: RET
Password for ATLANT/public:: ? RET

Directory ATLANT::/usr2/users/public/
README              bin                 printf.notes        filen
logfile1            logfile2
%
```

4.3.2 Using an Alias

As a shortcut to typing the node ID and access-control information, you can specify an "alias" node name. An alias node name is an alphanumeric string of one or more characters that you type in place of a node ID and any access-control information defined for it.

For example, you could have an alias "boo" that stands for user **tom** with password **secrets** on node BOSTON. With an alias like "boo," you do not have to specify any access-control information on the command line.

To use an alias in a command line, type the alias followed by the double colon (::) and the file specification. For example:

```
% dcat boo::'userdsk:[tom]memo.txt;3' RET
```

You can define an alias node name for any node on your network by creating a **.nodes** file in your home directory. Use the following format for your entries in the **.nodes** file:

alias=node-id[*/login-name*[*/password*]]

You cannot use spaces or tab characters in any of the fields. The following example shows sample **.nodes** file entries:

```
b=boston
w=boston/root/xyzkoroijt
boo=boston/tom/secrets
```

NOTE

Set up the protections on your **.nodes** file so that only the owner can read the file or write to it. Do this to prevent unauthorized access to your passwords.

4.3.3 Using Proxy Access

Using proxy access is another way to access a remote node without supplying access-control information. Although DECnet–ULTRIX systems support proxy access, not all DECnet systems do. Check with the manager of any non-ULTRIX system to find out if it does.

Before you can use proxy access, the system manager for the remote node must set up a proxy account for you. If you are going to have access to more than one proxy account from the same node and log-in name, indicate which proxy account is the default.

To use your default proxy account, enter the command without any access-control information. For example:

```
% dcat kokomo::farms.dat RET
```

To use an account other than your default proxy account, append the account login name to the node followed by a slash (/). The slash indicates that you are not supplying a password. For example:

```
% dcat kokomo/henry/::farms.dat RET
```

The *DECnet–ULTRIX Network Management* manual contains a full discussion of proxy access and instructions for defining proxy file entries.

Chapter 5

Working with Files on Remote DECnet Nodes

This chapter tells you how to:

- View directories on remote DECnet nodes
- Display and concatenate files from remote DECnet nodes
- Copy files to and from remote DECnet nodes
- Convert file names during file transfer
- Delete files on remote DECnet nodes

This chapter introduces the four DECnet–ULTRIX commands (**dls**, **dcat**, **dcp**, and **drm**) that you use to work with files on remote DECnet systems. Each command is illustrated with examples. For more information about each command, see Chapter 6 or the on-line manual pages.

Note that you can pipe or redirect input and outputfrom the **dls**, **dcat**, and **dcp** commands, according to standard ULTRIX conventions.

NOTE

Each sample command line in this chapter shows access-control information. Whenever you use the commands described in this chapter, you have to specify access-control information, unless you meet one of these criteria:

- You have defined an alias for the remote node.
- You have a proxy account on the remote node.

See Section 4.3 for a discussion about access-control information.

5.1 Viewing Remote Directories

The **dls** command displays the directories of a remote ULTRIX or non-ULTRIX node. The command displays the output on your terminal screen by default.

The syntax for the **dls** command is as follows:

dls [*options...*] *filespec* RET

For a complete description of the **dls** command, see Chapter 6.

5.1.1 On ULTRIX Remote Nodes

The following example displays **brightstar's** home directory on the remote DECnet–ULTRIX node BRAGG. The password is **starlet**.

The command line includes two options, **-a** and **-l**. The **-a** option causes all the files in the directory to be listed, including those whose names begin with a period. The **-l** option produces the long format, which includes the directory's **protection level**, creation date (or last modified date, for ULTRIX systems), size, and owner.

```
% dls -a -l bragg/brightstar/starlet:: RET

Directory WBRAGG::/usr/users/brightstar/
.                drwxrwxr-x   03-AUG-89 12:54:14      512   brightstar
..               drwxr-xr-x   29-JUL-89 15:09:39     2048   root
.cshrc           -rwxr-xr-x   05-MAY-89 19:03:53      281   brightstar
.forward         -rw-r--r--   05-MAY-89 20:12:49       16   brightstar
.login           -rwxr-xr-x   05-MAY-89 19:03:54      234   brightstar
.profile         -rwxr-xr-x   05-MAY-89 19:03:53      138   brightstar
log              -rw-r--r--   03-AUG-89 12:43:09      311   brightstar
text             -rw-r--r--   02-AUG-89 10:25:30      442   brightstar

8 files in 1 directory
%
```

5.1.2 On Non-ULTRIX Remote Nodes

The following example lists the files in directory [MANGO] on disk DRA2: on the DECnet–VAX node DAVIS. The access-control information includes the user name **s_wolf** and the password **quirk**.

```
% dls davis/s_wolf/quirk::'dra2:[mango]' RET

Directory DAVIS::DRA2:[MANGO]
FRUITY.TXT;2          FRUITS.LIS;6          TROPICAL.EXE;1
%
```

5.2 Displaying and Concatenating Remote Files

The **dcat** command displays the contents of a remote file and (by default) directs the output to your terminal screen, and also concatenates the contents of more than one file, following standard ULTRIX conventions.

The syntax for the **dcat** command is as follows:

dcat [*options...*] *filespec...* RET

For a complete description of the **dcat** command, see Chapter 6.

5.2.1 Displaying Remote Files on the Screen

You can use the **dcat** command to display remote files on your screen. In the following example, the command displays the contents of the file FRUITS.LIS in directory [MANGO] on disk DRA2: on the remote DECnet–VAX node DAVIS. The access-control information includes the user name **s_wolf** and the password **quirk**.

```
% dcat davis/s_wolf/quirk::'dra2:[mango]fruits.lis' RET
```

5.2.2 Concatenating Remote Files

You can also use the **dcat** command to concatenate multiple files. In the following example, all the files with the extension .TXT from the directory [ADOBE] on the remote DECnet–VAX node NAVAHO are concatenated and redirected into the local file **textfiles**. The access-control information includes the user name SANDY and the password BEACH.

```
% dcat navaho/sandy/beach::'[adobe]*.txt' > textfiles RET
```

5.3 Copying Files Between Systems

The **dcp** command lets you copy ASCII text and binary image files to and from remote DECnet nodes. You can copy files:

- From local to remote node
- From remote to local node
- From remote to remote node

The syntax for the **dcp** command is as follows:

dcp [*options...*] *input output* RET

For a complete description of the **dcp** command, see Chapter 6.

If you copy non-ULTRIX files to an ULTRIX system, you lose the non-ULTRIX attributes associated with those files.

5.3.1 From Local to Remote Node

In the following example, the command copies the local file **renee** to a new file RENEE.LIS on the remote VMS node WOODS in the directory [PETS.RACCOON]. The access-control information includes the user name TOMAS and the password TOM.

```
% dcp renee woods/tomas/tom::'[pets.raccoon]renee.lis' RET
```

5.3.2 From Remote to Local Node

The following command copies the file FARM.LIS from the remote DECnet–VAX node DAVIS to the file **farm** on the local DECnet–ULTRIX node. The access-control information includes the user name EVELYN and the password SECRET.

```
% dcp davis/evelyn/secret::'dra2:[mango]farm.lis' farm RET
```

The following command uses the **-i** option to copy, in image file mode, all the data files with the extension .DAT from the directory [HERO.HELIX.DATA] of HERO's account on the remote VMS node ONYX to the local directory **/usr/src/data**. The access-control information includes the user name HERO and the password MAGIC.

```
% dcp -i onyx/hero/magic::'[hero.helix.data]*.dat' /usr/src/data RET
```

The following command uses the **-r** option to copy all of the files in **~uucp** on the remote ULTRIX node KIMONO to the local directory. The access-control information includes the user name **larry** and the password **newcar**.

```
% dcp -r kimono/larry/newcar::~uucp . RET
```

5.3.3 From Remote to Remote Node

The following command copies the file [DON]NUM.TXT to the file [SUE.DATA]02.TXT. There is no access-control information in this example, because the aliases **red** and **blue** replace the full access-control information string. (For more information on using an alias, see Section 4.3.2.)

```
% dcp red::'user$55:[don]num.txt' blue::'user$22:[sue.data]02.txt' RET
```

5.4 Converting File Names During File Transfer

Non-ULTRIX DECnet systems do not follow the ULTRIX file-naming scheme. Therefore, when you copy a file from non-ULTRIX remote nodes to ULTRIX nodes, you may need to convert the file name. You can do this by using the **dcp -c** command option. Another way to solve this problem is to explicitly specify, on the command line, a name for the destination file.

By default, when you transfer a file, the file name changes (uppercase characters convert to lowercase, and the version number disappears) to match ULTRIX file-naming conventions. In the following example, the remote VMS file COSTS.TXT;3 is copied to the current directory on the local ULTRIX node. The access-control information is user name ANNA and password TOURIST:

```
% dcp venice/anna/tourist::'toni:[boats]costs.txt;3' . RET
```

This file appears as **costs.txt** in the local directory.

In the following example, the file name is not automatically converted during transfer because an output file name, **costs**, is specified in the command line:

```
% dcp venice/anna/tourist::'toni:[boats]costs.txt;3' costs RET
```

5.4.1 dcp -c Option Flags

The **-c** option flags control the format of the converted file output. By default, whenever you use the **dcp** command, the **ultrix** flags are in effect: **lower**, **nodollar**, **nosemicolon**, and **noversion**.

The value for each flag is defined as follows:

ultrix (default)	sets all **-c** flags (**lower**, **nodollar**, **nosemicolon**, and **noversion**)
lower	converts uppercase to lowercase
nodollar	converts '$' to underscore '_'
nosemicolon	converts ';' to '.'
noversion	strips off version numbers

Each flag also has a corresponding negative value:

none	clears all **-c** flags
nolower	does not convert uppercase to lowercase

dollar	does not convert '$' to underscore '_'
semicolon	does not convert ';' to '.'
version	does not strip off version numbers

NOTE

The **nosemicolon** flag is redundant when paired with the **noversion** flag. For example, if you specify **nosemicolon** and **noversion** when you transfer the file COSTS.TXT;3 to your local system, the semicolon is changed to a period, but that period is stripped off along with the version number.

5.4.2 Setting -c Option Flags

Use the **-c** option flags to change the way file names are converted during file transfer. Follow these guidelines:

- You can replace the **ultrix** default flags by adding the **setenv DCP_CNAMES** command to your **.login** file. For example:

```
setenv DCP_CNAMES nodollar,noversion
```

NOTE

Because the ULTRIX system is case sensitive, you must enter **DCP_CNAMES** in uppercase.

- You can add to the default (or **DCP_CNAMES**) flags by including **-c** option flags in the command line. For example:

```
% dcp -cnolower woods::'[pets.racoon]renee.lis;22' . RET
```

Note that the alias **woods** is used in this example in place of a complete access-control information string.

5.5 Deleting Remote Files

The **drm** command lets you delete a single file, multiple files, or an entire directory of files from a remote DECnet node. Of course, you need the appropriate access rights to delete any remote files and directories you specify.

The syntax for the **drm** command is as follows:

drm [*options...*] *filespec* RET

For a complete description of the **drm** command, see Chapter 6.

5.5.1 A Single File

In the following example, the **drm** command deletes the file FARM.LIS from the remote DECnet–VAX node DAVIS. The access-control information includes the user name S_WOLF and the password QUIRK:

```
% drm davis/s_wolf/quirk::'dra2:[mango]farm.lis' RET
```

5.5.2 Multiple Files

The **-r** option (recursive delete) for **drm** deletes all of the files and subdirectories from the directory **/usr/keith** on the remote DECnet–ULTRIX node IAMOK. The access-control information includes the user name **keith** and the password **partridge**. For example:

```
% drm -r iamok/keith/partridge::/usr/keith RET
```

5.5.3 All Files of a Single File Type

The following example deletes all the files with the .RNO extension from user WHITE'S account on the remote DECnet–VAX node ONYX. Because the files are in the home directory, no directory is specified.

```
% drm onyx/white::'*.rno' RET
Password for onyx/white:: ? snow RET  (not echoed)
%
```

Chapter 6

DECnet–ULTRIX Command Summary

This section describes each DECnet–ULTRIX user command in detail. The commands appear in alphabetical order and follow the graphic conventions set down in the Preface. Table 6–1 summarizes the functions of the DECnet–ULTRIX user commands.

Table 6–1: DECnet–ULTRIX Command Functions

Command	Function
dcat	Types the contents of remote files.
dcp	Copies files between DECnet systems.
diogin	Provides a virtual terminal connection to remote DECnet nodes.
dls	Lists the contents of a remote directory.
drm	Deletes remote files.
mail	Sends messages and files to remote DECnet users.

Note that the command descriptions do not discuss error messages. For a complete list of all possible error messages and a description of each message, see Appendix A.

The DECnet–ULTRIX command descriptions also appear on-line in **dcat(1dn)**, **dcp(1dn)**, **diogin(1dn)**, **dls(1dn)**, and **drm(1dn)**. In addition, all error messages are described in **errors(1dn)**.

dcat(1dn)

NAME

dcat — type the contents of remote files

SYNTAX

dcat [**-v**] *filespec...*

where

-v logs the names of the files being typed to standard error.

filespec is a file specification for one or more remote files. The format for a file specification varies with each Digital operating system.

You can specify wildcard characters. If you want the target node instead of the local shell to interpret a string of wildcard characters, you must enclose the string in quotation marks. See the *DECnet–ULTRIX Use* manual for file specifications and wildcard characters.

DESCRIPTION

The **dcat** command reads remote files and displays them on the standard output. The command displays the files in the order that you list them.

EXAMPLES

The following example displays, to standard output, the contents of the file FRUITS.LIS in the directory [MANGO] on disk DRA2: on the remote DECnet–VAX node DAVIS. Note that no access control information is given, indicating one of these possibilities: the file is world-readable, you defined an alias for the remote node, or you have a proxy account on the remote node.

```
% dcat davis::'dra2:[mango]fruits.lis'
```

The following command redirects all the files with the extension .TXT from the directory [ADOBE] on the remote DECnet–VAX node NAVAHO into the local file **lefile.txt**. The string **/adobe/secret** is the access-control information that NAVAHO uses to verify remote access.

```
% dcat navaho/adobe/secret::'[adobe]*.txt' > lefile.txt
```

SEE ALSO

errors(1dn)

dcp(1dn)

NAME

dcp — copy files between DECnet nodes

SYNTAX

dcp [*options...*] *input output*

where

-A appends the input file or files to a specified output file. Note that the output file must already exist; this option does not create an output file.

-P prints the files at the default printer on the remote system.

-S submits remote output files for execution. On ULTRIX systems, the **-S** option submits output files to the shell and creates a log file in the log-in directory that has the name *filename*.log, where *filename* is the name of the specified output file.

-a copies files in ASCII record mode. ASCII mode transfers perform the necessary format conversions between heterogeneous systems. ASCII mode is the default when you copy to and from non-ULTRIX systems.

-c converts the input file name from a non-ULTRIX system to a name with an ULTRIX format. This conversion happens by default whenever you use **dcp** to copy a file from a non-ULTRIX system (specifically, a VMS, RSX, or MS-DOS system) to an ULTRIX system.

Using the -c option, you can also specify one or more of the following flags to customize how file names are converted:

ultrix (default)	sets all **-c** flags (**lower**, **nodollar**, **nosemicolon**, and **noversion**)
lower	converts uppercase to lowercase
nodollar	converts $ to underscore _
nosemicolon	converts ; to .
noversion	strips off version numbers
none	clears all **-c** flags
nolower	does not convert uppercase to lowercase
dollar	does not convert $ to underscore _
semicolon	does not convert ; to .
version	does not strip off version numbers

The **-c** option has no effect unless the output file is in a local directory; you cannot use this option when copying files to remote directories.

-I copies files in image mode. This option is useful for copying nonprintable data files between homogeneous systems. Image mode transfers are generally faster than ASCII mode transfers but do not perform data format conversions between heterogeneous systems. Image mode is the default when you copy between ULTRIX systems.

-r copies all of the files in a directory. Also copies all subdirectories. The input and output names you specify must be directory names. Note that the top directory to which you are copying must already exist; **dcp -r** does not create it. However, this option does create all the subdirectories if they do not already exist on the node to which you are copying files. This option is valid only between DECnet–ULTRIX systems.

-v logs the names of the files being copied to standard error.

input is one or more input files or directory specifications. The format for an input file specification varies with the operating system on which the file is located. If there are multiple input files, separate each with a blank space.

You can specify a dash (–) in place of an input file specification, directing **dcp** to read from standard input until it reaches end-of-file (EOF).

You can specify wildcard characters. If you want the target node, instead of the local shell, to interpret a string of wildcard characters, you must enclose the string in quotation marks. See the *DECnet–ULTRIX Use* manual for more information.

output is the output file or directory to which **dcp** copies the input files. The format for an output file or directory specification varies with the operating system on which the output file is created. See the *DECnet–ULTRIX Use* manual for a description of all DECnet file specifications and wildcard characters.

When you copy input files to a directory, the output files retain the input file names and syntax unless you use the **-c** option. However, if you are copying a non-ULTRIX file to an ULTRIX system, the file name is automatically converted to match ULTRIX file-naming conventions.

You can use a dash (–) in place of the output file specification to direct the files to standard output.

DESCRIPTION

The **dcp** command copies files between DECnet nodes. You can copy both ASCII text and binary image files. Note that non-ULTRIX files with additional attributes lose those attributes when copied to an ULTRIX system.

When you use **dcp** to copy a file to another DECnet–ULTRIX system, you need not specify a mode of transfer because image mode is the default transfer mode between ULTRIX systems. For non-ULTRIX systems, you need to specify a mode of transfer only for image files.

File protection modes are preserved when you copy files between DECnet–ULTRIX systems. With non-DECnet–ULTRIX systems, the output file protection modes are defined by the remote system's file protection defaults.

When you copy files from a non-ULTRIX system to an ULTRIX directory without using the file-name conversion option, the output file name retains the format of the input file name. (By default, however, a file copied from a VMS system is automatically converted to ULTRIX file-naming conventions.) In other cases, you can use the **-c** option to convert file names.

RESTRICTION

You cannot use the **-c** option to convert file names when copying files between ULTRIX systems.

EXAMPLES

The following command copies the local file **farm.3** to the directory [MANGO] on device DRA2: on the remote DECnet–VAX node DAVIS; the command names the new file FARM.LIS. The access control information is **/mango/fruity**.

```
% dcp farm.3 davis/mango/fruity::'dra2:[mango]farm.lis'
```

The following command copies the file FARM.LIS from the remote DECnet–VAX node DAVIS to the local DECnet–ULTRIX node. By default, the file name will be converted to match ULTRIX file-naming conventions (lowercase, without dollar signs, semicolons, or version numbers).

```
% dcp davis::'dra2:[mango]farm.lis' .
```

The following command also copies the file FARM.LIS from the remote DECnet–VAX node DAVIS to the local DECnet–ULTRIX node. However, the output file name is not converted to lowercase because the user specifies the **-cnolower** flag.

```
% dcp -cnolower davis::'dra2:[mango]farm.lis' .
```

SEE ALSO

errors(1dn)

dlogin(1dn)

NAME

dlogin — log on to remote DECnet nodes

SYNTAX

dlogin *node* [**-ec**] [**-l** *logfile*]

where

node	is the DECnet node name or DECnet node address of the remote node to which you are connecting.
-ec	specifies an escape character for interrupting your remote session and temporarily returning control to your local node. The variable *c* is the character you define; any character is valid. The default escape character is the tilde (~).
-l *logfile*	logs your **dlogin** session to the file specified by the variable *logfile*.

DESCRIPTION

The **dlogin** utility lets you log on to remote DECnet nodes and access the resources of these operating systems.

This utility uses a protocol called the Digital Network Architecture (DNA) command terminal protocol (**CTERM**). With **dlogin**, you can connect to any DECnet node that supports CTERM. DECnet–ULTRIX nodes support both incoming and outgoing virtual terminal connections.

The **dlogin** utility lets you execute commands on your local node after you have started a remote session on another node. Whenever you want, you can switch back and forth between your local session and your **dlogin** session with the **dlogin** escape character.

To temporarily return control to your local node, at the remote system's prompt, type the **dlogin** escape character followed by RET. The default escape character is the tilde (~). You get a **local command>** prompt. At the **local command>** prompt, you can execute commands on your local node. To resume your **dlogin** session on the remote node, press RET at the **local command>** prompt.

To use the escape character without entering **local command mode**, type the escape character once, followed by any character except RET. You can also type the escape character twice, and it echoes to your screen once. For example, to send the **cd ~** command to the remote system, type **cd ~ ~** and press RET.

The **dlogin** utility offers help. For help on special commands that control your **dlogin** session, type a question mark (?) at the **local command>** prompt. The **dlogin** menu is displayed.

To end your **dlogin** session, type the remote node's logout command. To end a **dlogin** session after you have connected to a remote node but before you have logged on, type ~ RET and then issue the **exit** command.

RESTRICTION

CTERM is not supported on VMS versions before V4.0 or on DECnet–11M–PLUS versions before V3.0.

EXAMPLE

In the following example, user Irene logs on to the remote VMS node BACON and issues the SHOW USERS command:

```
% dlogin bacon

Username:  IRENE
Password:  GOODNIGHT    (not echoed)

                Welcome to VAX/VMS V5.2 on node BACON

$ SHOW USERS
```

SEE ALSO

errors(1dn)

dls(1dn)

NAME

dls — list the contents of a remote directory

SYNTAX

dls [**-1**] [**-C**] [**-a**] [**-l**] *filespec*

where

-1	formats the listing in a single-column format, which is the default when the standard output is not a terminal. The **-1** option is ignored if you use it with the **-l** option.
-C	formats the listing in a multicolumn format, which is the default when the standard output is a terminal. The **-C** option is ignored if you use it with the **-l** option.
-a	lists all the files in a remote ULTRIX directory, including the names that begin with a period (.). If you omit the **-a** option, file names that begin with a period are not listed.
-l	produces a list in long format. For each file specification, the **-l** option lists the file name, the protection settings, the creation date (or last modified date for ULTRIX systems), and the size in bytes.
filespec	is a file specification for a remote directory or file. The format for a file specification varies with each operating system. You can specify wildcard characters. If you want the target node instead of the local shell to interpret a string of wildcard characters, you must enclose the string in quotation marks. See the *DECnet–ULTRIX Use* manual for more information about file specifications and wildcard characters.

DESCRIPTION

The **dls** command lists all the file names in the specified remote directory or lists individual files. If you do not specify a directory or file, **dls** uses the directory name indicated by the access-control information.

All the file names in the directory are listed, unless you specify individual files. For each file name, **dls** repeats the name and any other information you request, for example, protection settings or creation dates.

Output from **dls -l** has designated characters that represent the protection setting. Displays from ULTRIX systems have 10 such characters, and **dls -l** displays from non-ULTRIX DECnet systems have 12 of these characters; for example:

```
DECnet-ULTRIX Node          DECnet-VAX Node
-rwxrw-r--                  ---rwxrwxrwx
```

For ULTRIX systems, the first character can be any of the following:

d indicates a directory

b indicates a block-type special file

c indicates a character-type special file

s indicates a socket

- indicates a regular file

The next nine characters represent the protection levels for the file's owner, group, and other (world), in that order, consisting of three characters each. Within each level, the three modes are represented by the characters **r**, **w**, and **x**, which are defined as follows (for a directory, execute permission implies permission to search the directory):

r indicates read permission

w indicates write permission

x indicates execute permission

– indicates that no permission has been set

The group-execute permission character is **s** if a file has the set-group-id bit set. Likewise, the user-execute permission character is **s** if a file has the set-user-id bit set.

The last character of an ULTRIX protection setting (normally **x** or **–**) is **t** if the sticky bit of the file mode is on. See the description of **chmod(1)** in the *ULTRIX Reference Pages* for the meaning of this mode. Some non-ULTRIX systems use different mapping schemes for file protection. These systems map their file protection schemes into the syntax used by **dls**. For example, an MS-DOS system does not have the concept of protections.

EXAMPLES

In the following example, the user specifies the remote DECnet–ULTRIX node ATLANT, the remote user name **jones**, and the password **mysecret**. Because this information is included in the file specification, the remote node can verify access:

```
% dls atlant/jones/mysecret:: RET
```

Using the long format, the following command lists the file name TEST.DAT, its protection level, creation date (or last modified date for ULTRIX systems), size, and owner. TEST.DAT is located on the remote DECnet–RSX node NAVAHO; note that the standard ULTRIX use of **>** *filename* redirects the information to the file **info.tes** on the local node:

```
% dls -l navaho::'[312,42]test.dat' > info.tes
```

SEE ALSO

errors(1dn)

drm(1dn)

NAME

drm — delete remote files

SYNTAX

drm [**-r**] *filespec*

where

-r	deletes all of the files from a directory and the directory itself.
filespec	is a complete file specification for a remote file or directory. The format for a file specification varies with each Digital operating system. You can specify wildcard characters. If you want the target node instead of the local shell to interpret a string of wildcard characters, you must enclose the string in quotation marks. See the *DECnet–ULTRIX Use* manual for more information about file specifications and wildcard characters.

DESCRIPTION

The **drm** command deletes one or more files from a remote system. The command can delete entire ULTRIX directories, but not non-ULTRIX directories.

RESTRICTION

If you specify the name of a directory, the **-r** option deletes all of the files in that directory, all of the files in all of the subdirectories, and both the specified directory and all existing subdirectories. This option is valid only between DECnet–ULTRIX systems.

EXAMPLES

This command deletes the file FARM.LIS from remote DECnet–VAX node DAVIS:

```
% drm davis::'dra2:[mango]farm.lis'
```

This command uses the **-r** option to delete all of the files in the directory **olddata** on the remote DECnet–ULTRIX node ATHENS. Note that if this command did not include a directory name, **drm** would use the access-control information **/george/seablue** and the effect would be to delete all of the files and all of the directories in account **george**:

```
% drm -r athens/george/seablue::olddata
```

SEE ALSO

errors(1dn)

mail

NAME

mail — send mail to DECnet users and receive mail from them

SYNTAX

mail *nodename*::*username*

mail *username*@*nodename*.dnet

where

nodename	is the name of the remote node where the user to whom you are sending mail resides.
username	is the name of the user to whom you are sending mail.
.dnet	is the Internet psuedodomain name.

DESCRIPTION

This command summary is not available on line. For on-line help, refer to **mail(1)** and **malladdr(7)**.

The **mail** command lets you send and receive mail to and from remote DECnet users. All of the flags, commands, and rules associated with ULTRIX **mail** are valid.

You can choose DECnet addressing syntax (*nodename*::*username*) or Internet addressing syntax (*username*@*nodename*.dnet). Note that the Internet addressing syntax is provided merely as a convenient interface for Internet users. DECnet–ULTRIX actually converts this Internet address syntax to the DECnet syntax before sending the mail out.

SEE ALSO

mail(1), malladdr(7)

Appendix A

Error Messages

This appendix describes all the possible error messages from **dcat**, **dcp**, **dlogin**, **dls**, and **drm** commands. Each error falls into one of the following categories:

- Connect errors
- File Access errors

Note that DECnet–ULTRIX error messages can also include ULTRIX errors and that descriptions of DECnet–ULTRIX error messages also appear on-line in **errors(1dn)**.

A.1 Connect Errors

Connect errors document failures to establish a DECnet network connection by either the local or the remote system.

Connect failed, Connection rejected by object

The network connection was rejected by the remote object.

Connect failed, Insufficient network resources

The network connection was rejected because of insufficient network resources on either the local or the remote node.

Connect failed, Unrecognized node name

The network connection could not be established because the local node does not know about the remote node.

Connect failed, Remote node shut down

The network connection could not be established because the remote node was shut down.

Connect failed, Unrecognized object

The remote node did not recognize the object. For more information, contact the remote node system manager.

Connect failed, Invalid object name format

The remote node did not understand the object name format of the connect request.

Connect failed, Object too busy

The network connection was rejected by the network partner because the remote object was too busy handling other clients.

Connect failed, Invalid node name format

The network connection could not be established because the format of the node name was incorrect. A node name is invalid if it contains illegal characters or is too long (node names can be up to 6 alphanumeric characters in length).

Connect failed, Local node is not on

The network connection could not be established because the network on the local node shut down.

Connect failed, Access control rejected

The network connection was rejected because the network partner could not successfully validate the access-control information it received. For example, you gave no proxy access and no default access exists.

Connect failed, No response from object

The network connection could not be established because the remote object did not respond. The remote object either responded too slowly or terminated abnormally.

Connect failed, Node unreachable

The network connection could not be established because no path currently exists to the remote node.

A.2 File-Access Errors

File-access errors document all error conditions other than DECnet connect errors.

Bad format DAP message received

An incompatibility in the Data Access Protocol (**DAP** version numbers or lower DECnet layers resulted in the losing and/or corrupting of the (DAP) message.

DAP error code *(macro:micro)*

where

macro	is the DAP macro error code
macro	is the DAP micro error code

This message appears when no other specific error message can be provided. For an explanation, see the DAP error code in **/usr/lib/dnet_shared/dap.errors**.

DAP message received out of order

An incompatibility in the DAP version numbers or lower DECnet layers resulted in losing and/or corrupting the DAP message.

Data type not supported

You attempted to copy a file whose data type is not supported by either the local or the remote system.

Device is write locked

You attempted to write to a file on a device that was write protected.

Device not in system

The device you specified is not known to the remote system.

Directory is full

The output file you specified cannot be created because there is no room available in the specified directory.

Error in file name

The file name you specified does not conform to the syntax of the remote system. See Appendix B for a definition of all DECnet file-name specifications.

File is locked

The file you are attempting to access is locked by the remote file system. This error can be caused by a remote system disallowing concurrent reading and writing of a file or by two users simultaneously attempting to write to the same file.

File organization not supported

You attempted to copy a file whose file organization was not sequential (DECnet–ULTRIX systems support only sequential files).

Link to partner broken

The DECnet network connection to the remote system was broken. This error can result when communication is no longer possible with the remote node.

Network operation failed

An operation to the network failed at the remote system.

Node name format error

The node name you specified was invalid; that is, the name contained illegal characters or was too long. See Appendix B for a definition of all DECnet node-name specifications.

No such device

The device you specified does not exist on the remote system.

No such file

The file you specified does not exist on the remote system.

Operation not supported locally

You attempted to perform an operation that is not supported by your local DECnet–ULTRIX system. For example, you cannot use **dls** to list a local directory.

Record attributes not supported

You attempted to copy a file whose record attributes are not supported by either the local or the remote system.

Record format not supported

You attempted to copy a file whose record format is not supported by either the local or the remote system.

Record too big for user's buffer

You attempted to copy a file that contained a record that is larger than **dcp**'s or the remote **fal**'s internal buffer. This error is frequently the result of an attempt to transfer a non-**ASCII** file in ASCII mode (which is the default transfer mode to non-ULTRIX systems).

Unspecified access error

An error occurred at the remote system in accessing a file. For an explanation, see the DAP error code in **/usr/lib/dnet_shared/dap.errors**.

Unsupported operation

The remote system does not support the operation you requested.

Appendix B

DECnet File Specifications

This appendix defines the syntax of file specifications for these DECnet systems:

- DECnet–ULTRIX
- DECnet–VAX
- DECnet–RSX
- DECnet–IAS
- DECnet/E
- DECnet-10
- DECnet-20
- DECnet–RT
- DECnet–DOS

The examples on the following pages show the different file specifications.

B.1 DECnet–ULTRIX File Specification

A DECnet–ULTRIX file specification has the following format:

node::path/filename

where

node	is the name or address of a DECnet–ULTRIX node.
path	is a list of directories, separated by slashes, that lead to the file name. – If *path* starts with a slash, this slash is the first character of the file specification, indicating that the path starts from the root file system. – If *path* starts with ~user, then ~user translates into the home directory of user on the remote ULTRIX system. – If the file specification does not begin with a slash, and no ~user part is specified, the file specification is relative to the home directory of the account through which access was granted.
filename	is an alphanumeric string of up to 255 characters that identifies a file.
*, ?, [ ], { }	A DECnet–ULTRIX file specification can also include any of these wildcard characters.

where

*	matches zero or more characters anywhere within a file name.
?	matches exactly one character.
[set]	matches exactly one character from a set. Set members can be enumerated (for example, [1234]) and/or contain ranges (for example, [0-4]) where the ASCII order is used.
{tokens}	matches one string from a list of ASCII strings, separated by commas, which can be a portion of the file name. For example, {abc,def}* will match any file name that begins with abc or def. A token can contain other wildcard characters.

EXAMPLES

1. The following examples are valid ULTRIX file specifications:

```
nashua
Mail/inbox/34
/usr/etc/fal
~austin/balloons
*
program.[ch]
record-{beatles,turtles,coasters}=top[123]0.19??
```

2. The following command copies the file **spring** from one DECnet–ULTRIX node to another. The receiving node is JUNEAU. Notice that you need not enclose the path and file names in quotation marks when you copy files between DECnet–ULTRIX nodes.

```
% dcp spring juneau::~jones/flowers RET
```

3. This example also copies the file **flowers** from one DECnet–ULTRIX node to another, but includes access-control information **/jones/market** with the remote node name QUINCY. Because the full path name and file name are not spelled out, the home directory for account **jones** is used.

```
% dcp flowers quincy/jones/market:: RET
```

4. This command copies the file FLOWERS.TXT from a DECnet–VAX node to the file **flowers** on the DECnet–ULTRIX node BOSTON. Note that the access-control information (**"james haymarket"**) and the file name are inside quotation marks so that DECnet–VAX passes the information to BOSTON as typed.

```
$ COPY FLOWERS.TXT BOSTON"james haymarket"::"flowers" RET
```

B.2 DECnet–VAX File Specification

A DECnet–VAX file specification has the following format:

node::*dev*:[*directory*]*filename.typ;ver*

where

node	is the name or address of a DECnet–VAX node.
dev	is a device on the VMS system, such as DRB2: or USER$22:.
directory	is a directory name, such as [HART.MEMOS] or [.MEMOS].
filename	is a string that names the file. On VMS V3.x systems, *filename* is an alphanumeric string of up to 9 characters. On VMS V4.*x* and V5.0 systems, *filename* is a string of up to 39 characters. Valid characters are alphanumerics, the underscore (_), the dollar sign ($), and the dash (–).
typ	is a character string that identifies the file type. On VMS V3.*x* systems, *typ* is an alphanumeric string of up to 3 characters. On VMS V4.*x* systems, *typ* is a string of up to 39 characters. Valid characters are alphanumerics, the period (.), the dollar sign ($), and the dash (–).
ver	is the file version number. The version number is a decimal number between 1 and 32767. Multiple versions of a file can exist; the latest version of a file is the one with the highest version number.

EXAMPLES

1. The following command copies the file **rivers** from a DECnet–ULTRIX node to the DECnet–VAX node VENICE. The new copy is placed in the directory [OSCAR.WATER] and is named RIVERS.TXT. Notice that the VMS file specification is enclosed within quotation marks so that VENICE, instead of the local shell, interprets this specification.

   ```
   % dcp rivers venice::'dba0:[oscar.water]rivers.txt' RET
   ```

2. This example also copies the file **rivers** from a DECnet–ULTRIX node to the DECnet–VAX node VENICE, but includes access-control information **/oscar/boats** with the node name. Also notice that because the file name is not specified, the file is copied to Oscar's default directory.

   ```
   % dcp rivers venice/oscar/boats:: RET
   ```

3. This DECnet–VAX command copies the file FLOWERS.TXT from a VMS node to the file **flowers** on the DECnet–ULTRIX node UTICA. Note that the information within quotation marks is in lowercase so that DECnet passes this information to UTICA in lowercase.

   ```
   $ COPY FLOWERS.TXT UTICA"mark upstate"::"flowers" RET
   ```

B.3 DECnet–RSX and DECnet–IAS File Specifications

DECnet–RSX and DECnet–IAS file specifications have the following format:

node::*dev*:[*ufd*]*filename.typ*;*ver*

node	is the name or address of a DECnet–RSX or DECnet–IAS node.
dev	is a device on the remote system, such as DB1:.
ufd	is the user file directory, which is an octal user identification code (uic), such as [312,42] or a named directory. The uic can range from [1,1] to [377,377]. RSX-11M-PLUS and Micro/RSX systems also support named directories such as [SNOW] and [SMITH].
filename	is an alphanumeric string of up to 9 characters that names the file.
typ	is an alphanumeric string of up to 3 characters that identifies the file type.
ver	is the file version number. The version number is a decimal number between 1 and 32767. Multiple versions of a file can exist; the latest version of a file is the one with the highest version number.

EXAMPLES

1. This command copies the file **pacific** from a DECnet–ULTRIX node to the DECnet–RSX node RAPA. The DECnet–RSX file specification is enclosed within quotation marks so that RAPA, instead of the local shell, interprets this specification.

```
% dcp pacific rapa::'db10:[50,377]pacific.txt' RET
```

2. This example also copies the file **pacific** from a DECnet–ULTRIX node to the DECnet–RSX node RAPA, but includes access-control information **/50,377/island** with the node name. Because the file name is not specified, the file is copied to the default directory for user 50,377.

```
% dcp pacific rapa/50,377/island:: RET
```

3. This command copies the file FLOWERS.TXT from a DECnet–RSX node to **/usr/tmp/flowers** on the DECnet–ULTRIX node UTICA. Note that the information within quotation marks is typed in lowercase so that DECnet passes this information to UTICA in lowercase.

```
$ NFT UTICA"mark research"::"/usr/tmp/flowers" = FLOWERS.TXT RET
```

B.4 DECnet/E File Specification

A DECnet/E file specification has the following format:

node::*dev*:[*ppn*]*filename.typ*

where

node	is the name or address of a DECnet/E node. You can append optional access-control information to the node name.
dev	is a device on the RSTS system, such as DK1:.
ppn	is a project programmer number, such as [314,42].
filename	is an alphanumeric string of up to 6 characters that names the file.
typ	is an alphanumeric string of up to 3 characters that identifies the file type.

EXAMPLES

1. The following command copies the file **flowers** from a DECnet–ULTRIX node to the DECnet/E node DAVIS. The DECnet/E file specification is enclosed within quotation marks so that DAVIS, instead of the local shell, interprets this specification.

```
% dcp flowers davis::'dm0:[50,25]test1.txt' RET
```

2. This example also copies the file **flowers** from a DECnet–ULTRIX node to the DECnet/E node DAVIS, but includes access-control information **/50,25/test** with the node name. Because the output file specification is not specified, the file is copied to the default directory for user 50,25.

```
% dcp flowers davis/50,25/test:: RET
```

3. This DECnet/E command copies the file TEST1.TXT from a DECnet/E node to **/users/jones/flowers** on the DECnet–ULTRIX node BOSTON. Note that the information within quotation marks is in lowercase type so that DECnet passes this information to BOSTON in lowercase.

```
$ COPY TEST1.TXT BOSTON"jones secret"::"~jones/flowers" RET
```

B.5 DECnet-10 File Specification

The file specification for a TOPS-10 operating system has the following format:

node::*dev*:[*ufd*] *filename.ext*[*p,pn*]<*prot*>

A complete file specification in an ANF-10 network has the following format:

node dev:filename.ext[*p,pn*]<*prot*>

or

logical name:filename.ext[*p,pn*]<*prot*>)

where

node	is the name or address of a DECnet-10 node. You can append optional access-control information to the node name.
node_dev	is a node name in the ANF-10 network, such as 1024_DSK0:.
dev	is a device on the TOPS-10 system, such as DSKC:.
ufd	is the user file directory.
filename	is an alphanumeric string of up to 6 characters that names the file.
ext	is an alphanumeric string of up to 3 characters, which is the file-name extension.
p,pn	is a project programmer number, such as [27,5117].
prot	is the file-access protection, which consists of up to 3 octal digits.

EXAMPLES

1. The following command copies the file **flowers** from a DECnet–ULTRIX node to a DECnet-10 node. The DECnet-10 file specification must be enclosed within quotation marks.

```
% dcp flowers atlant::'dba0:flowers.txt[52,879]' RET
```

2. This example also copies the file **flowers** from a DECnet–ULTRIX node to a DECnet-10 node, but includes access-control information **/52,879/secret** with the node name. Because the file name is not specified, the file is copied to the default directory for user 52,879.

```
% dcp flowers davis/52,879/secret:: RET
```

B.6 DECnet-20 File Specification

The file specification for a TOPS-20 operating system has the following format:

node::dev:<directory>filename.typ.gen;att

where

node	is the name or address of a DECnet-20 node.
dev	is a device on the TOPS-20 system, usually a file structure, such as SNARK:.
directory	is a directory name, such as <LONDONTOWN>.
filename	is an alphanumeric string of up to 39 characters that names the file.
typ	is an alphanumeric string of up to 39 characters that identifies the file type.
gen	is a generation or file version number.
att	is a file-access attribute.

EXAMPLES

1. The following command copies the file **snark** from a DECnet–ULTRIX node to the DECnet-20 node LONDON. The DECnet-20 file specification is enclosed within quotation marks so that LONDON, instead of the local shell, interprets this specification.

```
% dcp snark london::'dba0:<judy>garden.dat' RET
```

2. This example also copies the file **snark** from a DECnet–ULTRIX node to the DECnet-20 node LONDON, but includes access-control information **/judy/secret** with the node name. Because the output file is not specified, the file is copied to Judy's default directory.

```
% dcp snark london/judy/secret:: RET
```

B.7 DECnet-RT File Specification

An RT-11 file specification has the following format:

node::dev:filename.typ

where

node	is the name or address of a DECnet-RT node. You can append optional access-control information to the node name.
dev	is a device name on an RT-11 system, such as RK0:.
filename	is an alphanumeric string of up to 6 characters that names the file.
typ	is an alphanumeric string of up to 3 characters that identifies the file type.

EXAMPLES

1. The following command copies the file **daisy** from a DECnet–ULTRIX node to the DECnet-RT node RAGES. The DECnet-RT file specification is enclosed within quotation marks so that RAGES, instead of the local shell, interprets this specification.

```
% dcp daisy rages::rk0:test1.txt RET
```

2. This command also copies the file **daisy** from a DECnet–ULTRIX node to the DECnet-RT node RAGES, but specifies the access-control information **/sam/sam** for Sam's account.

```
% dcp daisy rages/sam/sam::test1.txt RET
```

B.8 DECnet–DOS File Specifications

A DECnet–DOS file specification has the following format:

node::dev:path\filename.typ

where

node	is the name or address of a DECnet–DOS node.
dev	is the drive name.
path	is the path name.
filename	is an alphanumeric file name of up to 8 characters.
typ	is an alphabetic file extension of up to 3 characters.

EXAMPLE

The following command copies the file **bear** from a DECnet–ULTRIX node to the DECnet–DOS node OZARK. The DECnet–DOS file specification is enclosed within quotation marks so that OZARK, instead of the local shell, interprets this specification.

```
% dcp bear ozark::'a:test1.txt'  RET
```

Glossary

This section defines the terms and concepts you must understand to read this book. This book assumes you are familiar with ULTRIX.

access-control information
Information used by the system to control access to system resources. Access control is the process of screening inbound connect requests and verifying them against a local system account file. Access control is optional. Access-control information consists of a user name, password, and account.

account
The allocation of system resources to each user. A user must have an account to use the system. Each user has a separate account, identified by a special account number and password.

alias
A short, meaningful name that replaces all, or some, of the access-control information you supply to a node.

ASCII file
A file in ASCII (American Standard Code for Information Exchange) format.

bidirectional gateway
The DECnet–Internet Gateway, which is network software that acts as a bridge between Internet and DECnet systems. The Gateway can support communication in two directions: from DECnet to Internet and from Internet to DECnet.

bidirectional gateway functions
The functions available through the DECnet–Internet Gateway, namely, remote login, mail exchange, and file access and transfer.

binary file
A file in binary (image mode) format.

case-sensitive
Refers to the system's ability to distinguish between uppercase (A-Z) and lowercase (a-z) letters.

connect errors
A type of error message which indicates that either the local or remote system failed to establish a DECnet network connection.

CTERM (Command Terminal Protocol)

The Digital Network Architecture (DNA) command terminal protocol. With **dlogin**, you can connect to any DECnet node that supports CTERM.

DAP (Data Access Protocol)

In the network application layer of the Digital Network Architecture (DNA), the protocol used for remote file access and transfer.

dcat (DECnet-ULTRIX concatenate)

The command that displays the contents of remote files.

dcp (DECnet-ULTRIX copy)

The command that copies files between DECnet systems.

dcp -c option flags

Flags you can use with the **dcp -c** option, namely, **ultrix**, **none**, **lower**, **nolower**, **nodollar**, **dollar**, **nosemicolon**, **semicolon**, **noversion**, and **version**. These flags can be combined to control how the **-c** option converts the file name.

DECnet-Internet Gateway

Digital Equipment Corporation's bidirectional gateway network software that acts as a bridge between Internet and DECnet systems. The Gateway is capable of supporting communication in two directions, from DECnet to Internet and from Internet to DECnet.

DECnet mail address

A syntax for addressing mail to remote DECnet users, namely, *username*::*node*.

DECnet node

Any node running DECnet software so that it can communicate with other nodes in the DECnet.

directory

A group of files stored on a disk. A user file directory is a file that briefly catalogs a set of files stored on tape or disk. The directory may include information such as the name, type, and version number of each file.

dlogin

The command that provides DECnet–ULTRIX end users with a virtual terminal connection to remote DECnet nodes.

dlogin escape character

The character you use to signal your local node to interrupt your **dlogin** session and return control to your local node. By default, the tilde (~) is the escape character.

dlogin session

The login session you start on a remote DECnet node using the **dlogin** command. The **dlogin** session lasts until you log off the remote DECnet node.

dls (DECnet-ULTRIX list)

The command that lists the contents of a remote DECnet directory.

dnet

The artificial domain name you include in the mail recipient's address when you use the optional Internet addressing syntax to send mail to remote DECnet users. Also called the "Internet psuedodomain name."

drm (DECnet-ULTRIX remove)

The command that removes, or deletes, remote DECnet files and directories.

fg (DECnet-ULTRIX foreground)

The command that resumes a suspended **dlogin** session. Enter this command at the **local command>** prompt.

file-access errors

A type of error message that indicates all error conditions other than DECnet connect errors.

file name

The title assigned to identify a specific file.

file specification

The unique identification of a file that gives its physical location and an indication of its contents. Different systems require different file specifications; refer to Appendix B for a discussion of the file specifications required by the DECnet systems supported by DECnet–ULTRIX.

help

You can display a list of **dlogin** commands by typing a question mark (?) at the **local command>** prompt.

Internet

A collection of packet-switching networks interconnected by gateways, along with protocols that allow them to function logically as a single,large,virtual network.

Internet mail address

An optional syntax for addressing mail to remote DECnet users, namely, *username@node*.dnet. This syntax is provided as a convenient interface for users familiar with the Internet syntax; DECnet–ULTRIX converts this address to the standard DECnet mail address before sending the mail out.

Internet psuedodomain name

The artificial domain name (.dnet) you include in the mail recipient's address when you use the optional Internet addressing syntax to send mail to remote DECnet users. See Section 3.1.

local command mode

An interface to your local node that you can access during a **dlogin** session. Use the escape character to enter local command mode. In local command mode, you can issue any ULTRIX or DECnet–ULTRIX command you want to execute on your local node. You can also issue DECnet–ULTRIX commands to control your **dlogin** session.

local login session

The login session you start on your local node when you log on directly. You can use the **dlogin** escape character to interrupt your **dlogin** session at any time to return to your local login session.

local node

The node you can log on to directly.

log file

A file containing a record of your input and the system's responses during a **dlogin** session.

mail

The ULTRIX command that lets you exchange electronic mail messages with other DECnet users.

man pages (manual pages)

Actual pages from reference manuals that you can display on your screen. The DECnet-ULTRIX man pages are printed in the DECnet–ULTRIX Command Summary (Chapter 6) of this manual.

man

The ULTRIX command that displays on-line manual pages on your screen. Manual pages are also called "command reference pages."

metacharacters

A group of keyboard characters (not including letters or digits) that have special meaning either to the shell or to the ULTRIX system. To use a metacharacter without its special meaning, either enclose it within quotation marks or precede it with a backslash.

node

An individual computer system in a network that can communicate with other computer systems in the network. Also called "host" and "system."

non-ULTRIX DECnet node

A DECnet node that runs an operating system other than ULTRIX (for example, VMS).

password

A combination of characters that verifies your identity to the computer.

path

The list of directories between the root directory and another directory. Also called "directory path."

piping

The process of sending the output from one command directly to another for use as the later command's input. You use the vertical bar character (|) as a pipe between commands. Although piping is not discussed in this manual, DECnet–ULTRIX supports piping according to standard ULTRIX conventions.

protection levels

The settings in each file that indicate who may and may not access the file. The settings are *read*, *write*, and *execute* privileges, and the groups are *owner*, *group*, and *world*.

proxy access

Proxy access allows you to gain access to a remote node without supplying access-control information. Proxy access uses proxy accounts, which system managers can establish.

redirection

The process of writing output from a command to a file using the right angle bracket (>), or of reading input for a command from a file using the left angle bracket (<). Although redirection is not discussed in this manual, DECnet–ULTRIX supports redirection according to standard ULTRIX conventions.

remote login session

The login session you start when you log on to a remote node. When you use the **dlogin** command to initiate the remote login session, the session is also called the **dlogin** session.

remote node

Any node in the network that is not the user's local node; any node that the user cannot log on to directly.

tilde (~)

The keyboard character that is the default **dlogin** escape character. The tilde is also the ULTRIX symbol for your home directory.

ULTRIX shell

The command interpreter for Digital Equipment Corporation. The ULTRIX product is a licensed derivative of UNIX software.

user name

The name a user types on a terminal to log on to the system.

wildcard character

A symbol, such as an asterisk or a percent sign, used within or in place of a file name, file type, directory, or version number in a file specification to indicate "all" for the given field.

world-readable file

A file all users can access, including both local and remote system operators, system managers, and users.

Index

HOW TO ORDER ADDITIONAL DOCUMENTATION

DIRECT TELEPHONE ORDERS

In Continental USA
call 800–DIGITAL

In Canada
call 800–267–6215

In New Hampshire
Alaska or Hawaii
call 603–884–6660

In Puerto Rico
call 809–754–7575

ELECTRONIC ORDERS (U.S. ONLY)

Dial 800–DEC–DEMO with any VT100 or VT200
compatible terminal and a 1200 baud modem.
If you need assistance, call 1–800–DIGITAL.

DIRECT MAIL ORDERS (U.S. and Puerto Rico*)

DIGITAL EQUIPMENT CORPORATION
P.O. Box CS2008
Nashua, New Hampshire 03061

DIRECT MAIL ORDERS (Canada)

DIGITAL EQUIPMENT OF CANADA LTD.
940 Belfast Road
Ottawa, Ontario, Canada K1G 4C2
Attn: A&SG Business Manager

INTERNATIONAL

DIGITAL
EQUIPMENT CORPORATION
A&SG Business Manager
c/o Digital's local subsidiary
or approved distributor

Internal orders should be placed through the Software Distribution Center (SDC),
Digital Equipment Corporation, Westminster, Massachusetts 01473

*Any prepaid order from Puerto Rico must be placed
with the Local Digital Subsidiary:
809–754–7575 x2012

DECnet–ULTRIX
Use
AA–PB61A–TE

READER'S COMMENTS

What do you think of this manual? Your comments and suggestions will help us to improve the quality and usefulness of our publications.

Please rate this manual:

	Poor				Excellent
Accuracy	1	2	3	4	5
Readability	1	2	3	4	5
Examples	1	2	3	4	5
Organization	1	2	3	4	5
Completeness	1	2	3	4	5

Did you find errors in this manual? If so, please specify the error(s) and page number(s).

_____

_____

_____

_____

General comments:

_____

_____

_____

_____

Suggestions for improvement:

_____

_____

_____

_____

Name _____ Date _____

Title _____ Department _____

Company _____ Street _____

City _____ State/Country _____ Zip _____

DO NOT CUT – FOLD HERE AND TAPE

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY LABEL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

digital™

Networks and
Communications Publications
550 King Street
Littleton, MA 01460–1289

DO NOT CUT – FOLD HERE

CUT ON THIS LINE

digital

DECnet–ULTRIX

DECnet–Internet Gateway Use and Management

May 1990

This manual tells you how to install, use, and manage the DECnet–Internet Gateway. The manual is for both DECnet and Internet system users and managers.

Supersession/Update Information:	This is a revised manual.
Operating System and Version:	ULTRIX V4.0
Software Version:	DECnet–ULTRIX V4.0

digital™

Order Number: AA–JQ71C–TE

AA–JQ71C–TE
May 1990

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by Digital or its affiliated companies.

Restricted Rights: Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227–7013.

The following are trademarks of Digital Equipment Corporation:

DEC	PDP	VAX
DECnet	ULTRIX	VMS
DECUS	UNIBUS	digital™

UNIX is a registered trademark of AT&T in the USA and other countries.

This manual was produced by Networks and Communications Publications.

Contents

Preface

This manual contains directions for installing, using, and managing the DECnet–Internet Gateway software.

Intended Audience

This manual is intended for DECnet–Internet Gateway users and anyone responsible for installing or managing the Gateway.

Chapter 2 applies to Internet host end users; Chapter 3 applies to DECnet node end users. End users should be familiar with general file-transfer principles. They should also understand how to use any systems they will be logging on to remotely.

Chapter 4 applies to Gateway managers. Gateway managers need superuser (root) access and/or system administrator privileges and must be familiar with general network management principles.

Structure of This Manual

This manual contains four chapters:

Chapter 1	Introduces the DECnet–Internet Gateway functions and commands, and contains instructions for installing the DECnet–Internet Gateway as part of the DECnet–ULTRIX software installation procedure. (The *DECnet–ULTRIX Installation* contains complete instructions for installing the Gateway software.)
Chapter 2	Explains how to use the DECnet–Internet Gateway from an Internet host. The examples in this chapter assume that your Internet host is an ULTRIX system; however, your Internet host might be another Internet system that is based on Berkeley 4.2/4.3 BSD Transmission Control Protocol/Internet Protocol (TCP/IP) implementations.
Chapter 3	Explains how to use the DECnet–Internet Gateway from a DECnet node. The examples in this chapter assume that your DECnet node is a DECnet–VAX node; however, your DECnet node might be another DECnet system, such as DECnet-11M-PLUS or DECnet–DOS.
Chapter 4	Contains guidelines for managing the DECnet–Internet Gateway software.

Related Documents

To supplement this manual, see the following documents:

- *DECnet–ULTRIX Release Notes*

  This manual contains miscellaneous information and updates not included in the DECnet–ULTRIX documentation set.

- *DECnet–ULTRIX Installation*

  This manual describes procedures for installing, configuring, and testing a DECnet–ULTRIX node. The manual also lists the names of the files installed with DECnet–ULTRIX software and gives their path names.

- *DECnet–ULTRIX Use*

  This manual contains both tutorial and reference information on how DECnet–ULTRIX end users can log on to remote DECnet nodes, exchange mail with users on remote DECnet nodes, and work with files on remote DECnet nodes.

- *DECnet–ULTRIX Programming*

  This manual explains concepts and guidelines for application programming in the DECnet–ULTRIX environment. The manual also describes DECnet–ULTRIX system calls and subroutines, and shows DECnet–ULTRIX data structures and programming examples.

- *DECnet–ULTRIX Network Management*

  This manual describes procedures for managing the network, such as defining permanent and volatile databases, node identifications and addresses, and lines and circuits; enabling event logging; displaying network counter information; operating and controlling a DECnet–ULTRIX node; and testing the network operation.

- *DECnet–ULTRIX NCP Command Reference*

  This manual describes how to use the Network Control Program (**ncp**) to perform network management functions.

For a detailed description of the Digital Network Architecture (DNA), refer to the *DECnet Digital Network Architecture (Phase IV) General Description.*

Graphic Conventions

This manual uses the following conventions:

Convention	Meaning
special	In running text, commands, command options, parameters, system calls, subroutines, user names, file names, and directory names appear in this **special** type.
`example`	Indicates an example of system output or user input. System output is in black type; user input is in red type.
lowercase	If a command appears in lowercase type in a command format or in an example, you enter it in lowercase.
UPPERCASE	Node and Gateway names appear in uppercase. VMS commands also appear in uppercase.
italic	Italic type in command formats and system displays indicates a variable, for which either you or the system supply a value.
[ ]	Square brackets enclose optional data. You can use only one of the enclosed options. Do not type the brackets when you enter the command line.
key	Indicates a key on your keyboard. CTRL/*key* represents a CTRL key sequence, where you press the CTRL key at the same time as the specified key. Note that keyboard keys are represented by this symbol, <*key*>, on line.
%	The percent sign is the standard DECnet–ULTRIX system prompt.
#	The pound sign is the DECnet–ULTRIX superuser prompt.

All Ethernet addresses are hexadecimal; all other numbers are decimal unless otherwise noted.

Terminology

In this manual, "DECnet–RSX" stands for any of these DECnet products: DECnet–11M–PLUS, DECnet–Micro/RSX, DECnet–11S, DECnet–11M.

Also, "Gateway" stands for the DECnet–ULTRIX DECnet-Internet Gateway software.

Chapter 1

Introduction

This chapter describes the DECnet–Internet Gateway functions, some sample user commands, and installation.

DECnet–Internet Gateway software provides bidirectional access between DECnet systems (such as DECnet–VAX, DECnet–RSX, and DECnet–DOS) and Internet systems (such as those based on Berkeley 4.2/4.3 BSD TCP/IP implementations).

The Gateway software lets you:

- Log on to an Internet system from a DECnet system and vice versa
- Exchange mail between these systems
- Access files on both types of systems
- Transfer files between these systems

Systems that use the DECnet–Internet Gateway software do not have to run special software, and remote users do not have to establish accounts on the Gateway system.

1.1 Gateway Functions and Sample Commands

The Gateway supports a subset of DECnet and Internet commands. Table 1–1 shows the DECnet–VAX and Internet commands by function that the Gateway software supports.

Table 1–1: Commands Supported Through the DECnet–Internet Gateway

Gateway Function	DECnet–VAX (VMS) Command	Internet (UNIX) Command
Exchange mail	MAIL	mail
Log in	SET HOST	telnet
Work with Files		
Change directory	(No command)	ftp> cd
Display directory	DIRECTORY	ftp> ls
		ftp> dir
Show current directory	(No command)	ftp> pwd

(continued on next page)

Table 1–1 (Cont.): Commands Supported Through the DECnet–Internet Gateway

Gateway Function	DECnet–VAX (VMS) Command	Internet (UNIX) Command
Delete files	DELETE	ftp> delete
		ftp> mdelete
Display files	TYPE	ftp> get *file* -
		ftp> recv
Transfer files	COPY	ftp> recv
		ftp> get
		ftp> mget
		ftp> send
		ftp> put
		ftp> mput
	APPEND	ftp> append

The Internet commands are described in Chapter 2, and the DECnet-VAX commands are described in Chapter 3.

1.2 How to Use This Manual

If you log on to an Internet system (such as ULTRIX) and use the Gateway to access DECnet systems, see Chapter 2 for instructions.

If you log on to a DECnet system (such as a DECnet–VAX or DECnet–RSX system) and use the Gateway to access Internet systems, see Chapter 3 for instructions.

If you manage an Internet, DECnet, or Gateway host system, see Chapter 4 for instuctions.

1.3 Installing the Gateway

Before you install the DECnet–Internet Gateway software onto your DECnet–ULTRIX node, configure your system for Internet by choosing the ULTRIX Internet subset when you install the ULTRIX base software.

You can use the DECnet–ULTRIX installation procedure to install both the DECnet–ULTRIX base software and the DECnet–Internet Gateway. You can install these software subsets at the same time, or the base software first and the Gateway afterward.

You can install the DECnet–Internet Gateway as a unidirectional gateway, enabling gateway functionality in one direction only. For more information, see Section 4.1.

For a complete description of the installation procedure, see *DECnet–ULTRIX Installation*.

Chapter 2

Using the Gateway from an Internet Host

This chapter tells you how to perform these tasks while logged on to an Internet host:

- Log on to a remote DECnet node
- Exchange mail with a remote DECnet node
- Work with files on a remote DECnet node

Note that the examples in this chapter assume that your Internet host is an ULTRIX system. Your Internet host might be a non-Digital Internet system based on Berkeley 4.2/4.3 BSD TCP/IP implementations. In that case, you might use different commands to perform the tasks described in this chapter.

2.1 Sample Internet Commands

Table 2–1 lists common Internet commands you can use through the Gateway.

Table 2–1: Sample Internet Commands and Their Functions

Command	Function
ftp	Starts an **ftp** session.
ftp> append	Copies a local file to the end of a remote file.
ftp> ascii	Sets the file transfer type to ASCII mode.
ftp> binary	Sets the file transfer type to binary image mode.
ftp> bye	Closes an **ftp** session.
ftp> cd	Changes the current remote directory.
ftp> close	Terminates a connection to a remote node.
ftp> delete	Deletes a single remote file.
ftp> dir	Lists the contents (in long form) of a remote directory.
ftp> disconnect	Terminates a connection to a remote node.
ftp> get	Copies a remote file to the local directory. (Same as **ftp> recv**.)
ftp> ls	Displays the names of files in the remote directory.
ftp> mdelete	Deletes multiple remote files.
ftp> mget	Copies multiple remote files to the local directory.
ftp> mput	Copies multiple local files to the remote directory.

(continued on next page)

Table 2–1 (Cont.): Sample Internet Commands and Their Functions

Command	Function
ftp> open	Starts a connection to a remote node.
ftp> put	Copies a local file to the remote directory. (Same as ftp> send.)
ftp> pwd	Displays the full path name of the current remote directory.
ftp> quit	Closes the ftp session.
ftp> recv	Copies a remote file to the local directory. (Same as ftp> get.)
ftp> send	Copies a local file to the remote directory. (Same as ftp> put.)
ftp> type	Shows current file transfer type.
mail	Sends mail to remote DECnet users.
telnet	Provides log on to remote DECnet nodes.

2.2 Logging On to a Remote DECnet Node

You can use the Gateway to log on to a remote DECnet node. Once you have logged on, or established a remote login session with the remote node, you can use programs running on that node. This section shows you how to begin and end a remote login session.

You must have an account on the remote node before you can log on.

2.2.1 Beginning a Remote Login Session

To log on to a DECnet node through the Gateway, type telnet and the Gateway host name at the system prompt. In the following example, the Gateway host is BOSTON:

```
% telnet boston RET
```

When the Gateway's login prompt appears, type the name of the DECnet node you want to log on to, followed by a double colon. In the following example, the target DECnet node is LYONS.

```
boston login: lyons:: RET
```

The DECnet node then prompts you for the DECnet user name and password set up for you on that system. In this example, the user is Dube. The password does not appear, or echo, when you type it.

```
Username: DUBE RET
Password: secret RET  (not echoed)

    Welcome to VAX/VMS version V5.2 on node LYONS
Last interactive login on Thursday, 4-SEP-1990 13:13
```

User Dube is now logged in to the remote DECnet node and can run programs, work with files, and perform other tasks. For information about the tasks and activities you can perform on remote DECnet nodes, see the documentation for those nodes.

NOTE

If you specify the user name or password incorrectly, you must start over. If you try to finish logging on without breaking the connection,

you will be logging on to the Gateway node instead of the target node or host.

2.2.2 Ending a Remote Login Session

You end your remote login session by logging off the remote node. To log off the remote DECnet node, simply type the remote node's logout command. Logout commands vary; for example, the command is LOGOUT for DECnet–VAX systems and **bye** for DECnet–RSX systems. If you do not know the logout command for the remote operating system you are connected to, see that system's documentation.

The following example shows a complete remote login session. User Dube logs on to the DECnet node LYONS, deletes a file, and types the VMS LOGOUT command to end the remote login session:

```
% telnet boston   RET
Trying...
Connected to boston.
Escape character is '^]'.
boston login: lyons::   RET

Username: DUBE   RET
Password: secret  RET   (not echoed)

    Welcome to VAX/VMS version V5.2 on node LYONS
Last interactive login on Thursday, 4-SEP-1990 13:13

$ DELETE TESTFILE.TXT;*   RET
$ LOGOUT   RET
  DUBE          logged out at 1-SEP-1990 13:18:30.29
dlogin -- session terminated
Connection closed by foreign host.
%
```

2.3 Exchanging Mail

The **mail** utility lets you communicate with DECnet users across the Internet. To send mail to a DECnet user, enter the **mail** command followed by the user's mail address. The mail address includes this information:

- Recipient's DECnet user name
- Recipient's DECnet node name
- DECnet communication domain symbol (dnet)
- Gateway host name

Use the following format:

mail *username%node.dnet@gate*

where

username	is the user assigned to the target account.
node	is the target DECnet node name.
dnet	is the DECnet psuedo-domain name.
gate	is the DECnet–Internet Gateway host name.

The "%" and "@" symbols are separators in the mail address format.

The following example shows the Internet user Susan sending a message to the DECnet user Simone on node PARIS. The Gateway host is BOSTON.

```
% mail simone%paris.dnet@boston RET
Subject: Today's teleconference postponed RET
RET
The team teleconference scheduled for today is postponed to RET
Tuesday, May 29th, at 2 PM. RET
RET
                                    Susan RET
CTRL/D
CC:ben RET
%
```

2.4 Working with Files on Remote DECnet Nodes

You can use **ftp** commands to work with files and directories on remote DECnet nodes. The Gateway supports twenty **ftp** commands, which are discussed in the following sections and in the *ULTRIX Reference Pages, Section 1*.

To use **ftp**, first start an **ftp** session. You start a session by invoking the **ftp** program. The session lasts until you end it. During an **ftp** session, you see the **ftp>** prompt.

You also establish a connection with the remote DECnet node that you want to access. Once connected, you can use **ftp** commands to manipulate files and directories on that node.

The following sections describe specific tasks you can perform using **ftp** commands.

2.4.1 Starting an ftp Session

To start an **ftp** session, simply type **ftp** at the system prompt. For example:

```
% ftp RET
ftp>
```

2.4.2 Ending an ftp Session

To end an **ftp** session, type **bye** or **quit** at the **ftp>** prompt. For example:

```
ftp> quit RET
%
```

2.4.3 Connecting to a Remote DECnet Node

To access files on a remote DECnet node using **ftp**, you first need to establish a connection to that node. Once you are connected to a node, you can use **ftp** commands to work with files and directories on that node.

You can connect to a node before or after you have started the **ftp** session. Perform one of the following steps:

- If you have not started the **ftp** session yet, type the **ftp** command and the Gateway name at the Internet system prompt. In this example, the Gateway system is BOSTON:

  ```
  % ftp boston RET
  ```

- If you have already started the **ftp** session, type the **open** command and the Gateway name at the **ftp>** prompt. In this example, the Gateway system is BOSTON:

```
ftp> open boston RET
```

When the Gateway prompts you for a user name, specify the name of the DECnet node to which you want to connect and your user name on that node. Enter the information in this format: *node*::*username*.

In this example, BOSTON is the Gateway host name, PARK is the target DECnet node name, and Renee is the DECnet user name.

```
Name (boston:renee): park::renee RET
```

Finally, enter the password at the Password prompt. The password does not appear on your screen, or echo, when you type it. For example:

```
Password (boston:park::renee): secret RET  (not echoed)
```

NOTE

You may not receive an error message if you enter your password incorrectly. The system simply does not execute your commands. Instead, it displays messages such as "Requested file action not taken," "Broken pipe," and "Directory unavailable."

Also, the message, "Access control rejected," may indicate that your DECnet node name/user name string may be too long for the **ftp** implementation you are using. (Some implementations set a 16-character name limit.) Section 4.3 tells you how to correct this incompatibility.

Here is an example of the complete connection process. Again, the Gateway is BOSTON, the DECnet node is PARK, the user is Renee, and the password is **secret**.

```
% ftp RET
ftp> open boston RET
Connected to boston.
220 boston FTP server (Version 4.1 Sun Aug 7 19:42:25 EDT 1990) ready.
Name (boston:renee): park::renee RET
Password (boston:park::renee): secret RET  (not echoed)
331 Password required for gateway access park::renee.
230 Access control info received.
ftp>
```

2.4.4 Disconnecting from a Remote DECnet Node

To end your connection to a remote DECnet node, type the **close** or **disconnect** command at the **ftp>** prompt. For example:

```
ftp> close RET
221 Goodbye.
ftp>
```

Note that the **close** command does not end the **ftp** session.

2.4.5 Viewing Remote Directories

While you are connected to a remote DECnet node, you can use directories on that node, as follows:

To display the name of the current remote directory, type the **pwd** command at the **ftp>** prompt. For example:

```
ftp> pwd RET
257 "[RENEE]" is current directory.
ftp>
```

To change the current remote directory, type the **cd** command followed by the name of the directory. Enter the complete path name of the remote directory in the syntax expected by the remote system. If you do not specify the directory name, the system prompts you for one. For example:

```
ftp> cd [renee.memos] RET
250 CWD command successful.
ftp>
```

or

```
ftp> cd RET
(remote-directory) [renee.memos] RET
250 CWD command successful.
ftp>
```

To return to the home directory after accessing a subdirectory in **ftp**, specify the full path name and the correct directory syntax with the **cd** command. You can also close and reopen your connection to return to the home directory.

To display the names of the remote files, type the **ls** command. For example:

```
ftp> ls RET
200 PORT command successful.
150 Opening data connection for dls (123.45.6).
BC.LN03;3
BC.TXT;12
STATUS.SDML;2
TIPS.TXT;
226 TRANSFER COMPLETE.
54 bytes received in 0.83 seconds (0.064 Kbytes/s)
ftp>
```

For an expanded display, type the **dir** command. For example:

```
ftp> dir RET
200 PORT command successful.
150 Opening data connection for dls (123.45.6).
BC.LN03;3         rwxrwxr-x--    14-MAR-90 12:55:10    29266  [430,501]
BC.TXT;12         rwxrwxr-x--    12-MAR-90 00:12:55     7718  [430,501]
STATUS.SDML;2     rwxrwxr-x--    19-APR-90 10:19:15     6534  [430,501]
TIPS.TXT;1        rwxrwxr-x--    22-DEC-89 07:53:59     2038  [430,501]
226 TRANSFER COMPLETE.
300 bytes received in 0.71 seconds (0.41 Kbytes/s)
ftp>
```

2.4.6 Displaying Remote Files

To display a remote file, type the **get** command followed by the name of the remote file and a hyphen. For example:

```
ftp> get team.txt - RET
200 PORT command successful.
150 Opening data connection for
    park/renee/password::team.txt (123.45.6)
This file lists all the reviewers for Tim's book:
Jim
Susan
Pat
William
Dan S.
Dan M.
Maryellen
226 Transfer complete.
remote: team.txt
111 bytes received in 0.15 seconds (0.72 Kbytes/s)
ftp>
```

The hyphen indicates that you want the file to be displayed. If you omit the hyphen, the system copies the remote file to your local directory, without displaying anything.

The **recv** command works exactly like **get**.

2.4.7 Setting the File Transfer Type

You can transfer, or copy, ASCII and binary files using **ftp** commands. Set the Gateway file transfer type to ASCII when you transfer ASCII files and to binary (image mode) when you transfer binary files. By default, the Gateway transfers files in ASCII.

You can display and set the file transfer type using the **ftp** commands shown in the following examples.

To display the current file transfer type, enter the **type** command. For example:

```
ftp> type RET
Using binary mode to transfer files.
ftp>
```

To set the file transfer type to ASCII, enter the **type ascii** or **ascii** command. For example:

```
ftp> type ascii RET
200 Type set to A.
ftp>
```

or

```
ftp> ascii RET
200 Type set to A.
ftp>
```

To set the file transfer type to binary image mode, type the **type binary** or **binary** command. For example:

```
ftp> type binary RET
200 Type set to I.
ftp>
```

or

```
ftp> binary  RET
200 Type set to I.
ftp>
```

2.4.8 Copying Files Between Systems

While you are connected to a remote DECnet node, you can copy files to and from that node. The following examples illustrate the different ways you can copy files with **ftp** commands.

To copy a remote file, type either the **get** or **recv** command, followed by the remote file name. If you want the copy to have a different name, also type the local file name. The **get** and **recv** commands are interchangeable.

In this example, a local file name is not specified, so **ftp** creates a local file with the same name as the remote file:

```
ftp> recv tasks.lis  RET
200 PORT command successful.
150 Opening data connection for
    park/renee/password::tasks.lis (123.45.6)
226 Transfer complete.
local: tasks.lis remote:tasks.lis
6010 bytes received in 0.8 seconds (7.3 Kbytes/s)
ftp>
```

In this example, a local file name is specified:

```
ftp> get tasks.lis tasks  RET
200 PORT command successful.
150 Opening data connection for
    park/renee/password::tasks.lis (123.45.6)
226 Transfer complete.
local: tasks remote:tasks.lis
6010 bytes received in 0.41 seconds (14 Kbytes/s)
ftp>
```

NOTE

If you use the **get** or **recv** command and type a hyphen instead of specifying a local file name for the file, the file is displayed without being copied.

To copy multiple remote files, type the **mget** command, followed by the remote file names. The system prompts you to accept or reject each file. Press RET for yes or type **n** for no. The following example uses the wildcard character (*) to copy all the files with the .txt extension:

```
ftp> mget *.txt RET
mget MEMO73.txt;1 ? RET
200 PORT command successful.
150 Opening data connection for
    park/renee/password::memo73.txt;1 (123.45.6)
226 Transfer complete.
local: MEMO73.TXT;1 remote:MEMO73.TXT;1
24000 bytes received in 1.1 seconds (21 Kbytes/s)
mget DOCTYPE.TXT;2 ? n RET
mget DOCTYPE.TXT;3 ? RET
200 PORT command successful.
150 Opening data connection for
    park/renee/password::doctype.txt;3 (123.45.6)
226 Transfer complete.
local: DOCTYPE.TXT;3 remote:DOCTYPE.TXT;3
9398 bytes received in 6.6 seconds (1.4 Kbytes/s)
ftp>
```

To copy a local file to a remote file, type either the **send** or **put** command, followed by the local file name and the remote file name (optional). The **send** and **put** commands are interchangeable. For example:

```
ftp> send team.txt newteam.txt RET
200 PORT command successful.
150 Opening data connection for
    park/renee/password::team.txt (123.45.6)
226 Transfer complete.
local: team.txt remote: newteam.txt
6010 bytes sent in 0.4 seconds (42 Kbytes/s)
ftp>
```

To copy multiple local files to a remote node, type the **mput** command, followed by the local file names. The system prompts you to accept or reject each file. Press RET for yes or type **n** for no.

The following example uses the wildcard character (*) to copy every file named **status**, no matter what its extension:

```
ftp> mput status.* RET
mput status.dat ? RET
200 PORT command successful.
150 Opening data connection for
    park/renee/password::status.dat (123.45.6)
226 Transfer complete.
local: status.dat remote: status.dat
6010 bytes sent in 0.1 seconds (50 Kbytes/s)
mput status.txt ? RET
200 PORT command successful.
150 Opening data connection for
    park/renee/password::status.txt (123.45.6)
226 Transfer complete.
local: status.txt remote: status.txt
6010 bytes sent in 0.14 seconds (42 Kbytes/s)
ftp>
```

To copy a local file to the end of a remote file, type the **append** command followed by the local file name and the remote file name. The following example appends the local file **scores** to the remote file **team.txt**:

```
ftp> append scores team.txt RET
200 PORT command successful.
150 Opening data connection for
    park/renee/password::team.txt (123.45.6)
226 Transfer complete.
local: scores remote: team.txt
9398 bytes sent in 0.59 seconds (16 Kbytes/s)
ftp>
```

You cannot use the **append** command to copy a remote file to the end of a local file.

2.4.9 Deleting Remote Files

To delete a single remote file, type the **delete** command followed by the filename. For example:

```
ftp> delete team.txt RET
250 DELE command successful.
ftp>
```

If you do not specify the file name after the delete command, the system prompts you for the file name. For example:

```
ftp> delete RET
(remote-file) team.txt RET
250 DELE command successful.
ftp>
```

To delete multiple remote files, type the **mdelete** command and the file names. The system prompts you to accept or reject each file. Press RET for yes or type **n** for no. For example:

```
ftp> mdelete memo.txt memo.ln03 memo.ps RET
mdelete MEMO.TXT;1? RET
250 DELE command successful.
mdelete MEMO.LN03? RET
250 DELE command successful.
mdelete MEMO.PS? RET
250 DELE command successful.
ftp>
```

You can use wildcards when you use the **mdelete** command. For example:

```
ftp> mdelete memo.* RET
mdelete memo.ln03 ? RET
250 DELE command successful.
mdelete memo.txt;10 ? n RET
mdelete memo.txt;9 ? RET
250 DELE command successful.
mdelete memo.txt ? RET
250 DELE command successful.
ftp>
```

Chapter 3

Using the Gateway from a DECnet Node

This chapter tells you how to perform these tasks while logged on to a DECnet node:

- Log on to an Internet host
- Exchange mail with an Internet host
- Work with files on an Internet host

Note that the examples in this chapter assume that your DECnet node is a VMS system. Your DECnet node might be another DECnet system, such as DECnet–VAX or DECnet–RSX. In that case, you might use different commands to perform the tasks described in this chapter.

3.1 Sample DECnet–VAX Commands

Table 3–1 lists common DECnet–VAX commands you can use through the Gateway.

Table 3–1: Sample DECnet–VAX Commands and Their Functions

Command	Function
APPEND	Copies a remote file to the end of a local file.
COPY	Transfers a file to or from an Internet host.
DELETE	Deletes a file.
DIRECTORY	Displays the file names in an Internet directory.
MAIL	Sends mail to remote Internet users.
SET HOST	Provides remote login to Internet systems.
TYPE	Displays the contents of a file.

3.2 Logging On to a Remote Internet Node

With the DECnet–Internet Gateway, you can use the VMS SET HOST utility to log in to an Internet host as an Internet user. Once you have logged on, or established a remote login session with the Internet host, you can use programs running on that host. This section shows you how to begin and end a remote login session.

You must have an account on the remote node before you can log on.

3.2.1 Beginning a Remote Login Session

To log on to an Internet host through the Gateway, type the SET HOST command and the Gateway host name at the system prompt. In this example, the Gateway is BOSTON:

```
$ SET HOST boston RET
```

When the Gateway node displays a login prompt, enter the target Internet host name and an exclamation point (!). Do not include spaces or tabs in this string.

```
login: lyons! RET
```

The Internet host then prompts you for access-control information. You can log in as usual by entering your user name and password. In this example, the user name is Renee. Your password does not appear on the screen as you type it.

```
lyons login: renee RET
Password: secret RET  (not echoed)

Last login: Thu Jun 28 11:57:49 from 1.0.0.6
4.3 BSD UNIX #0: Thu May 29 11:18:26 EDT 1990
```

User Renee is now logged in to the Internet host and can run programs, work with files, and perform other tasks. For information about the tasks and activities you can perform on the Internet host, refer to the documentation describing the host.

NOTE

If you specify the user name or password incorrectly, you must start over. If you try to finish logging in without breaking the connection, you will be logging in to the Gateway node instead of the target Internet host.

3.2.2 Ending a Remote Login Session

You end the remote login session by logging off the remote Internet node. To log off, simply type the remote node's logout command. Logout commands vary from system to system; your system may require CTRL/D or **logout**. If you do not know the logout command for the remote operating system you are working on, see that system's documentation.

The following example shows a complete remote login session. User Renee uses the Gateway node BOSTON to log on to the Internet host LYONS. Then Renee displays the **report.2** file and presses CTRL/D to end the login session:

```
$ SET HOST boston RET

Ultrix V4.0 (boston)

login: lyons! RET
Trying...
Connected to lyons.
Escape character is '^]'.
lyons login: renee RET
Password: secret RET  (not echoed)
Last login: Thu Jun 28 11:57:49 from 1.0.0.6
4.3 BSD UNIX  0: Thu May 29 11:18:26 EDT 1990

% cat report.2 RET
This report outlines the results of our last experiment.
.
.
.
```

```
% logout RET
$
```

3.3 Exchanging Mail

The MAIL utility lets you communicate with Internet users. To send mail to an Internet user, enter the MAIL command followed by the user's mail address. The mail address consists of the Gateway node name, the recipient's Internet host name, the recipient's Internet user name, and the communication domain (optional).

Use the following format:

MAIL *gate*::"*username@host*[.*DOMAIN*]"

where

gate	is the DECnet–Internet Gateway node name, which is a string of 1 to 6 alphanumeric characters.
username	is the user name assigned to the target Internet account.
host	is the target Internet node name.
DOMAIN	is the communication domain, which is optional. Each Internet host has a **sendmail** file for routing mail in a heterogeneous network. Each system manager can configure **sendmail** for different types of mail addressing. Whether or not the domain is required depends on how your network is configured. Some examples of domains are ARPA, UUCP, and LOCAL. For more information about **sendmail**, refer to the *ULTRIX Reference Pages, Section 8*, and the article entitled "SENDMAIL - An Internetwork Mail Router" in the *ULTRIX Supplementary Documents, Volume III*.

Here is a sample DECnet–VAX mail message that Dave is sending through the Gateway BOSTON to Sarah, a user on the Internet host TULSA:

```
$ MAIL RET

MAIL> send RET
To:     boston::"sarah@tulsa" RET
Subj:   Vacation RET

Enter your message below.  Press CTRL/Z when complete, or CTRL/C to quit:

I won't be able to attend the meeting next week because RET
I will be on vacation. RET

                                        Dave RET

CTRL/Z
Exit

MAIL> exit RET
$
```

3.4 Working with Files

From your DECnet system prompt, you can use five DECnet–VAX commands to work with files and directories on an Internet host: APPEND, COPY, DELETE, DIRECTORY, and TYPE. The following sections on Gateway-supported DECnet–VAX commands explain how to type command lines, which tasks you can perform with these commands, and how to shorten your file specifications.

3.4.1 Using DECnet–VAX Commands

When you use DECnet–VAX commands through the Gateway, you must specify the Gateway name, access-control information, and file-specification information. Use either of the following formats:

command gate"*inet*!*username password*"::"*filename*"

command gate"*username*@*inet password*"::"*filename*"

where

command	is a Gateway-supported DECnet command.
gate	is the DECnet–Internet Gateway node name, which is a string of up to 6 alphanumeric characters.
inet	is the target Internet host name contained in the access-control information.
username	is the user assigned to the target Internet account. This is part of the access-control information that you supply to the target node.
password	is the Internet user's password. This is part of the access-control information that you supply to the target node.
filename	is the string of alphanumeric characters that identifies the file. It can be an absolute specification, such as **/bin/login**, or a relative one, such as **directory/file2.c**. The file name is optional.

NOTE

Enclose the file name in quotation marks. Otherwise, VMS interprets the file name according to DCL command syntax, changing lowercase characters to uppercase and adding an extension and version number to the file name. The access-control information must also be set off in quotation marks.

3.4.2 Viewing Remote Directories

To list the file names in remote Internet directories, type the DIRECTORY command and specify the directory and files (optional) you want displayed. For example:

```
$ DIRECTORY boston"bean!a_lima topsecret":: RET

Directory BOSTON"bean!a_lima password"::

Message1              dnetcheck.rnd       doctype.txt          inventory
tasks.s               team.txt

Total of 6 files.
$
```

NOTE

This command lists only the file names within specific directories; any other information you receive may be incorrect.

3.4.3 Displaying Remote Files

To display remote Internet files, enter the TYPE command followed by the file name. For example:

```
$ TYPE boston"bean!a_lima topsecret"::"team.txt" RET
Here are the members of Bill's design team:
Jim, Susan, Pat, William, Maryellen, Dan S. and Dan M.
$
```

3.4.4 Copying Files Between Systems

To copy a remote file to your local system, type the COPY command followed by the remote file name and the local file name. For example:

```
$ COPY boston"bean!a_lima topsecret"::"dnetcheck.rnd" dnetcheck.txt RET
```

To copy a local file to a remote system, type the COPY command followed by the local file name and the remote file name. For example:

```
$ COPY easynotes.lis boston"bean!a_lima topsecret"::"easynotes.lis" RET
```

To copy the contents of a remote file to the end of a local file, type the APPEND command followed by the remote file name and the local file name. In the following example, the phone numbers in the remote file **phones.txt** are copied to the end of the local file **team.txt**:

```
$ APPEND boston"a_lima@bean topsecret"::"phones.txt" team.txt RET
```

3.4.5 Deleting Remote Files

To delete a remote file, type the DELETE command followed by the name of the remote file to be deleted. For example, both of the following command lines remove the file **report.1** from Kiko's account on the Internet host TOKYO. The password is **secret**; the Gateway node name is BOSTON.

```
$ DELETE boston"tokyo!kiko secret"::"report.1" RET
$ DELETE boston"kiko@tokyo secret"::"report.1" RET
```

Notice that the file name must be in lowercase type enclosed within quotation marks to ensure that the Internet host receives the file name in lowercase as required.

3.4.6 Using Shortcuts in File Specifications

To shorten the file specifications that you type, you can use wildcard characters and logical names.

3.4.6.1 Using Wildcard Characters

Both the Gateway and the VMS operating system support the asterisk (*) wildcard character. The asterisk replaces a string of alphanumeric characters.

When you want your DECnet node to interpret a wildcard character, just include it in the file specification. The following example copies all files ending with .c from a DECnet node to the target host NEWLONDON. Note that the file names will appear in VMS format (uppercase text with version numbers) at the destination.

```
$ COPY *.c boston"newlondon!sailor racetime":: RET
```

When you want the target Internet host rather than your DECnet node to interpret a wildcard character, enclose the file specification in quotes. For example, the following command removes all files ending with .c from Liana's account on the Internet host NEWYORK:

```
$ DELETE boston"newyork!liana sailing"::"*.c" RET
```

3.4.6.2 Using Logical Names

You can define a logical name to use in place of the Gateway node name, target host name, and access-control information in a file specification. Using a logical name can help improve accuracy and convenience.

Use the VMS DEFINE command to define a logical name as follows:

DEFINE *logical-name*"*gate*""*inet!username password*"":"

where

logical-name	is the new name that you assign to a portion of the file specification. This string can contain 1 to 255 characters, including alphanumerics, the dollar sign, or the underscore.
"gate""inet!username password"":"	is the Gateway node name, Internet host name, Internet user name, and password that you want to associate with the logical name. This string can contain 1 to 255 characters, including alphanumerics, the dollar sign, or the underscore. As this format shows, you enclose the string in quotation marks and use two sets of quotation marks ("") in the places where you want one quotation mark (") to appear.

NOTE

You cannot use the *username@inet* format when you define logical names.

For example, the following command assigns the logical name SIDNEY to the Gateway node name BOSTON, target host name SIDNEY, user name Charlie, and password **downunder**. Notice that the string includes the double colon that normally follows a node name in a DECnet–VAX file specification.

```
$ DEFINE sidney "boston""sidney!charlie downunder"":" RET
```

Include your logical name in your **login.com** file, so it is redefined every time you log in.

Once you have defined a logical name, you can use it in place of the file specification. In the following example, **sidney** replaces the **boston"sidney!charlie downunder"::** string.

```
$ COPY file.dat sidney RET
```

3.5 Special Considerations for Non-UNIX-Based Internet Systems

If you are transferring text files from a non-UNIX-based Internet system, watch for the following:

- A file you transfer appears on your system with no end-of-line terminators.
- The file transfer fails, and an error message indicates that the record was too large for your buffer.

These problems may indicate that the Gateway's default data transfer mode is inappropriate for transferring files from non-UNIX-based Internet systems. Contact your system administrator. For more information, see Section 4.4.

Chapter 4

Managing the Gateway

This chapter tells you how to manage the DECnet–Internet Gateway by performing these tasks:

- Controlling access to Gateway functions
- Keeping a log of connections to the Gateway

As this chapter also shows, you can manage these common Gateway situations:

- Handling length restrictions for access-control information
- Special considerations for non-UNIX-based Internet systems

You need superuser or system administrator privileges to perform these tasks.

4.1 Controlling Access to the Gateway

Your system can perform as a Gateway node only if the Gateway software has been enabled. If, when you were installing DECnet-ULTRIX, you configured your node to run the Gateway software, it was enabled at that time. The following sections describe how you can manually control file transfer and remote login for both DECnet and Internet networks.

NOTE

You do not have to enable the mail systems; both DECnet and Internet mail systems run in Gateway mode when DECnet–ULTRIX is installed.

If you enable access in both directions (from DECnet to Internet and from Internet to DECnet) the Gateway operates as a bidirectional gateway. You can also configure the Gateway as a unidirectional gateway by disabling either DECnet-to-Internet or Internet-to-DECnet access. With access in only one direction enabled, the Gateway functions as a unidirectional gateway. The following sections describe how to enable and disable access in both directions. Note that the DECnet–ULTRIX installation configures the Gateway, by default, as a bidirectional gateway.

4.1.1 Controlling Access from DECnet to Internet

You can control the use of your node as a DECnet Gateway for file transfer and remote login functions. (This task requires system administrator privileges.)

To enable or disable DECnet file transfer and remote login Gateway functions, use the **ncp set executor** command:

```
ncp set executor gateway access [ enabled  ]
                                [ disabled ]
```

The **enabled** option turns on DECnet Gateway access, and the **disabled** option turns it off. The following example turns on DECnet Gateway access:

```
$ ncp set executor gateway access enabled RET
```

To use file transfer, you must also define the Gateway user. When a Gateway node receives a file-access request, its DECnet object spawner tries to verify any access-control information that the request contains. However, since that information is for the destination Internet host, not for the Gateway node, the DECnet object spawner sets the privileges for the object (on the Gateway node) to those of the Gateway user.

Define the Gateway user with the following **ncp** command format:

ncp set executor gateway user *login-name*

where *login-name* specifies the default login name for the Gateway. To ensure system security, give the Gateway user **guest** privileges. For example, the following command defines the Gateway user as **guest**, because **guest** has limited user privileges.

```
% ncp set executor gateway user guest RET
```

4.1.2 Controlling Access from Internet to DECnet

You can control the use of your node as an Internet Gateway for file transfer functions or remote login functions or both. This task requires superuser privileges.

To enable or disable Internet file transfer and remote login functions, edit the file **/etc/inetd.conf**, then stop the **inetd** process and restart it. When you edit **/etc/inetd.conf**, find the lines that contain the following daemon names:

```
/etc/ftpd
/etc/telnetd
/etc/ftpd.gw
/etc/telnetd.gw
```

To turn on Gateway file transfer services, delete the pound sign (#) in front of the line containing **/etc/ftpd.gw**, and add a pound sign in front of the line containing **/etc/ftpd**. When the lines look like this, access is turned on:

```
#ftp      stream   tcp   nowait     /etc/ftpd      ftpd
 ftp      stream   tcp   nowait     /etc/ftpd.gw   ftpd
```

To turn on Gateway remote login services, delete the pound sign (#) in front of the line containing **/etc/telnetd.gw**. Then add a pound sign at the beginning of the line containing **/etc/telnetd**. For example:

```
#telnet   stream     tcp    nowait     /etc/telnetd     telnetd
 telnet   stream     tcp    nowait     /etc/telnetd.gw  telnetd
```

After you edit **/etc/inetd.conf**, stop the **/etc/inetd** process by using the **kill** command in the following format:

kill -9 *process-number*

In this example, the user finds the process number for **inetd**, kills the process, and restarts it:

```
# ps -ax | grep inetd  RET
  104 ?  I      0:02 /etc/inetd
 1789 18 S      0:00 grep inetd
# kill -9 104  RET
# /etc/inetd  RET
       [1] 1792
```

4.2 Keeping a Log of Connections to the Gateway

You can keep records of connections to the Gateway, including file-access and remote-login connections.

4.2.1 File-Access Connections

You can keep records of file-access connections in two ways: through the file **/usr/adm/wtmp**, which keeps track of **ftp** activity, and through the **syslog** function, which keeps track of File Access Listener (**fal**) activity.

To view the contents of **/usr/adm/wtmp**, enter the **last** command as follows:

```
% last gateway  RET
gateway    ftp        boston      Mon Aug  3 15:59 - 16:03  (00:04)
gateway    ttyp0      MONTRL      Mon Aug  3 15:54 - 15:59  (00:01)
gateway    ttyp0      atlant      Mon Aug  3 15:51 - 15:55  (00:01)
```

Each **ftp** Gateway entry in **/usr/adm/wtmp** lists **gateway** as the user name, **ftp** as the type of request, the name of the node that issued the request, the date and time, and the duration of the connection. In the following example, node BOSTON made the request and the connection lasted for 4 minutes:

```
gateway    ftp        boston      Mon Aug  3 15:59 - 16:03  (00:04)
```

Note that **/usr/adm/wtmp** also keeps track of remote login activity; for more information, refer to the following section.

For each **fal** connection, the **syslog** function records the process number, object name, type of access, node name, Gateway user name, target host, file name, and exit time. Each entry begins with the date, time, and local host. In this example, 1482 is the process number. The connection is from **art** on node MONTRL.

```
Aug 29 13:57:02 localhost: 1482 fal: DIRECTORY access from
MONTRL::ART, user=guest, to host=boston, filename=a.c
```

For more information about **syslog**, see **syslog(8)** in the *ULTRIX Reference Pages, Section 8*.

4.2.2 Remote-Login Connections

You can keep records of remote-login connections in the file **/usr/adm/wtmp**, which contains an entry for each **telnet** and **dlogin** connection.

Entries for **dlogin** and **telnet** in **/usr/adm/wtmp** list **gateway** as the user name, the terminal type (**ttyp***n*), the name of the node issuing the request, the time and date, and the duration of the connection. In this example, ttyp0 is the terminal, and MONTRL is the requesting node:

```
% last gateway RET
gateway   ttyp0      MONTRL      Mon Aug  3 15:54 - 15:55  (00:01)
```

When **dlogin** logs a request, the node name appears in uppercase, as in the previous example. When **telnet** logs the request, the node name appears in lowercase.

To view **/usr/adm/wtmp**, issue the **last** command.

4.3 Handling Length Restrictions for Access-Control Information

Some implementations of **ftp** limit access-control information to 16 characters. If your DECnet node name and user name exceed 16 characters (including the double colon), the remote DECnet node rejects the access.

Other **ftp** implementations limit passwords to 8 characters. In this case, you may receive an "Access control rejected" message if your DECnet password is longer than 8 characters.

Avoid violating these restrictions by establishing a new account on the DECnet node with a user name of 8 or fewer characters and a password of 8 or fewer characters. If this method is not possible, there are other solutions depending upon the **ftp** implementation you are using. Two possible solutions follow.

4.3.1 Establishing Connections Through .netrc Files

Some implementations support **ftp** connections through a file named **.netrc**. The **.netrc** file allows the user to specify the remote **ftp** system, user name, and password. The **ftp** utility reads the file and uses the information contained within it instead of prompting the user. This file must exist in the user's home directory and, if it contains a password, must be readable only by the user.

Entries in **.netrc** should have the following format:

```
machine gateway_name
login decnet_node_name::user_account_name
password user_account_password
```

For example, user **massachusetts_man** on the VMS system JEWEL with the password *massachusetts* would bypass login prompting when using the Gateway system FOCUS if the following entry existed in **.netrc** in his home directory:

```
machine focus
login jewel::massachusetts_man
password massachusetts
```

4.3.2 Using the quote Command

If your **ftp** implementation supports the use of the **quote** command, you can use it to bypass user and password size restrictions. However, there is no way to disable echoing of the password to the screen.

To use the **ftp quote** command, invoke **ftp** with autologin disabled (use the **-n** option). At the **ftp>** prompt, enter the **ftp** command name preceded by **quote**. To send the user name, enter **quote user** *user_account_name*. To send the password, enter **quote pass** *user_account_password*. The following example shows how to use the **quote** command:

```
% ftp -n focus RET
Connected to focus.
220 focus FTP server (Version 4.1 Tue Mar 1 16:47:05 EST 1990) ready.
ftp> quote user jewel::massachusets_man RET
331 Password required for gateway access jewel::massachusetts_man.
ftp> quote pass massachusetts RET
231 Access control info received.
ftp>
```

4.4 Special Considerations for Non-UNIX-Based Internet Systems

Users may encounter problems if they initiate file transfers with a non-UNIX-based Internet system that does not use a newline \n as the end-of-line terminator.

By default, the Gateway performs image mode transfers over the Internet connection and allows the DECnet connection to handle any data conversions. However, this default mode assumes that the newline character denotes end-of-line.

If a file transfer to a DECnet system using **copy** (or another DECnet file transfer command) results in a file with no end-of-line terminators or in an error message of "record too large for the user's buffer," it may be the result of the Gateway's default data transfer mode. This is true for text files only.

You can solve this problem by configuring the Gateway to force the Internet connection to preserve the data transfer mode. For example, if you perform an ASCII mode transfer over the Internet connection when the transfer mode for the DECnet connection is ASCII, you must set the environment variable **fal_inet** to **nonunix**.

Be aware that if you do force the Gateway connection to preserve the data transfer mode over the Internet connection, an incompatibility may arise in **ftp** implementations based on the Berkeley 4.2 BSD implementation.

As part of the ULTRIX V3.0 product, changes based on 4.3 BSD have been incorporated into some of the Internet applications. As a result, the same changes have been incorporated into **fal** for the Internet connection handling.

One of the incorporated changes based on 4.3 BSD is the handling of embedded carriage returns (not end-of-line indicators) in ASCII text files. Prior to 4.3 BSD, as well as in earlier versions of the ULTRIX product and the DECnet–Internet Gateway product, if an embedded carriage return was not immediately followed by a newline character, a null character was inserted into the data stream following the carriage return. On the receiving side, the null character was discarded before writing the data to a file.

The default mode for the Gateway follows 4.3 BSD conventions. That is, a null character is not inserted into the data stream if an embedded carriage return is found in a file, nor is a null character inserted into a data stream following a carriage return stripped.

If the Internet systems to which you are transferring files through DECnet-initiated commands are based on 4.2 BSD, and if you expect to transfer files that may contain embedded carriage returns, set the **fal** environment variable **fal_ascii** to **crnul**.

Note that this variable will have meaning only if you have also set **fal_inet** to **nonunix**. If not, the data transfer over the Internet connection occurs in binary mode, thus preventing data conversions.

Index

HOW TO ORDER ADDITIONAL DOCUMENTATION

DIRECT TELEPHONE ORDERS

In Continental USA
call 800–DIGITAL

In Canada
call 800–267–6215

In New Hampshire
Alaska or Hawaii
call 603–884–6660

In Puerto Rico
call 809–754–7575

ELECTRONIC ORDERS (U.S. ONLY)

Dial 800–DEC–DEMO with any VT100 or VT200
compatible terminal and a 1200 baud modem.
If you need assistance, call 1–800–DIGITAL.

DIRECT MAIL ORDERS (U.S. and Puerto Rico*)

DIGITAL EQUIPMENT CORPORATION
P.O. Box CS2008
Nashua, New Hampshire 03061

DIRECT MAIL ORDERS (Canada)

DIGITAL EQUIPMENT OF CANADA LTD.
940 Belfast Road
Ottawa, Ontario, Canada K1G 4C2
Attn: A&SG Business Manager

INTERNATIONAL

DIGITAL
EQUIPMENT CORPORATION
A&SG Business Manager
c/o Digital's local subsidiary
or approved distributor

Internal orders should be placed through the Software Distribution Center (SDC),
Digital Equipment Corporation, Westminster, Massachusetts 01473

*Any prepaid order from Puerto Rico must be placed
with the Local Digital Subsidiary:
809–754–7575 x2012

DECnet–ULTRIX DECnet–Internet
Gateway Use and Management
AA–JQ71C–TE

READER'S COMMENTS

What do you think of this manual? Your comments and suggestions will help us to improve the quality and usefulness of our publications.

Please rate this manual:

	Poor				Excellent
Accuracy	1	2	3	4	5
Readability	1	2	3	4	5
Examples	1	2	3	4	5
Organization	1	2	3	4	5
Completeness	1	2	3	4	5

Did you find errors in this manual? If so, please specify the error(s) and page number(s).

_____

_____

_____

_____

General comments:

_____

_____

_____

_____

Suggestions for improvement:

_____

_____

_____

_____

Name _____ Date _____

Title _____ Department _____

Company _____ Street _____

City _____ State/Country _____ Zip _____

DO NOT CUT – FOLD HERE AND TAPE

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY LABEL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

digital™

Networks and
Communications Publications
550 King Street
Littleton, MA 01460–1289

DO NOT CUT – FOLD HERE

CUT ON THIS LINE

digital

DECnet–ULTRIX

Programming

May 1990

This manual offers guidelines for application programming in the DECnet–ULTRIX environment, describes DECnet–ULTRIX system calls and subroutines, and shows DECnet–ULTRIX data structures and programming examples.

Supersession/Update Information:	This is a revised manual.
Operating System and Version:	ULTRIX V4.0
Software Version:	DECnet–ULTRIX V4.0

digital™

Order Number: AA–EA88D–TE

AA–EA88D–TE
May 1990

This manual was produced by Networks and Communications Publications.

Contents

Preface

The DECnet–ULTRIX product is layered software that runs on an ULTRIX system. With this software, an ULTRIX system functions as an end node in a DECnet network. The DECnet–ULTRIX product is an end-node implementation of Digital Network Architecture (DNA) Phase IV.

Manual Objectives

Using both tutorial and reference material, this manual show programmers how to write programs for client and server applications in the DECnet–ULTRIX environment.

Intended Audience

This manual is for programmers using DECnet–ULTRIX software to write network applications. The manual assumes the following:

- You are familiar with an editor, such as **vi** or **ed**.
- You have a working knowledge of the C programming language and experience writing system or network programs.
- You are familiar with the ULTRIX system, including its naming conventions, system commands, system calls, and subroutines.

Structure of This Manual

The *DECnet–ULTRIX Programming* manual is divided into two parts, five chapters, and two appendixes.

Part I introduces DECnet–ULTRIX programming concepts and guidelines for application programming in the DECnet–ULTRIX programming environment:

Chapter 1	Describes DECnet–ULTRIX programming concepts.
Chapter 2	Describes DECnet–ULTRIX programming tools and how they work in the DECnet–ULTRIX programming environment.
Chapter 3	Explains how to write programs for clients and server applications in the DECnet–ULTRIX programming environment.

Part II contains descriptions and other reference information about the DECnet–ULTRIX system calls and subroutines:

Chapter 4	Describes the DECnet–ULTRIX system calls.
Chapter 5	Describes the DECnet–ULTRIX subroutines.

The appendixes show DECnet data structures and programming examples.

Appendix A	Contains the DECnet–ULTRIX data structures.
Appendix B	Contains DECnet–ULTRIX programming examples.

Related Documents

To supplement the *DECnet–ULTRIX Programming* manual, refer to the following manuals:

- *DECnet–ULTRIX Release Notes*

  This document contains miscellaneous information and updates not included in other books in the DECnet–ULTRIX documentation set.

- *DECnet–ULTRIX DECnet–Internet Gateway Use and Management*

  This manual describes the DECnet–Internet Gateway and contains directions for installing, using, and managing, it.

- *DECnet–ULTRIX Network Management*

  This manual defines the DECnet–ULTRIX network databases and components. It describes the Network Control Program (**ncp**) and how it is used to configure, monitor, and test your network. Other topics include loopback testing, event logging, and instructions for displaying network information.

- *DECnet–ULTRIX NCP Command Reference*

  This reference manual describes the **ncp** commands used for defining, monitoring, and testing your network.

- *DECnet–ULTRIX Installation Guide*

  This manual describes procedures for installing a DECnet–ULTRIX node and testing it for proper operation. This manual also lists the DECnet–ULTRIX distribution files and the path names to which they are installed.

- *ULTRIX Guide to the Data Link Interface (DLI)*

  This manual describes procedures for using DLI to write application programs at the data link layer.

- *ULTRIX Introduction to Network Programming*

  This manual describes network programming concepts for programming in the ULTRIX environment.

To obtain a detailed description of DNA, refer to the *DECnet Digital Network Architecture (Phase IV), General Description.*

Graphic Conventions

This manual uses the following graphic conventions:

Convention	Meaning
special	Command options, system calls, subroutines, and data structures appear in **special type**.
command( )	Cross-references to specific command documentation include the section number in the reference manual where the commands are documented. For example: See the **socket(2dn)** system call. This indicates that you can find the material on the **socket** system call in Section 2dn of the reference pages.
`literal`	Indicates terms that are constant and must be typed just as they are presented.
[ ]	Square brackets indicate optional arguments. Do not type the brackets.
...	Horizontal ellipsis points indicate that the preceding item can be repeated one or more times. **NOTE** In examples, vertical ellipsis points represent either user input or system input that has been omitted to emphasize specific information.
lowercase/ UPPERCASE	Because DECnet–ULTRIX software is case-sensitive, you must type all literal input in the case shown. UPPERCASE is also used for the names of all DECnet nodes, including DECnet–ULTRIX nodes. This convention follows DECnet protocol, which names and recognizes all nodes in UPPERCASE. However, node names are not case-sensitive and need not be typed in the case shown.
`example`	Indicates an example of system output. System output is in black type; user input is in red type.
italics	Indicate a variable, for which either you or the system must specify a value.
%	The default user prompt in multiuser mode.
#	The default superuser prompt.
key	Indicates a key on your keyboard. CTRL/*key* represents a CONTROL key sequence, where you press the CONTROL key at the same time as the specified key.

Other conventions are as follows:

- All numbers are decimal unless otherwise noted.
- All Ethernet addresses are hexadecimal.

Part I

Overview

Chapter 1

Introduction to the DECnet–ULTRIX Programming Environment

This chapter introduces some of the DECnet programming concepts on which the DECnet–ULTRIX programming interface is based. All terms and concepts in this chapter are presented in the context of the DECnet–ULTRIX programming environment.

For more information about programming in the ULTRIX environment, see the *ULTRIX Network Programming Guide* and the article "A 4.2 BSD Interprocess Communication Primer," in the *ULTRIX Supplementary Documentation, Volume III*.

1.1 DECnet–ULTRIX Programming Interface

The DECnet–ULTRIX programming interface lets you write cooperating programs that exchange data over a DECnet network. The interface provides the following support:

Client–server communication. This is sometimes called task-to-task communication. The client application initiates a connection and requests services from the server application. The server application either accepts or rejects the request. Client-server communication lets DECnet–ULTRIX Phase IV applications communicate with remote Phase III and Phase IV DECnet applications through a socket-level programming interface.

DECnet and TCP/IP coexistence. DECnet protocols and Transmission Control Protocol/Internet Protocol (TCP/IP) coexist and can share system resources, including Ethernet and Digital Data Communications Message Protocol (DDCMP) hardware. You can modify most TCP/IP programs to use DECnet protocols, or DECnet programs to use TCP/IP protocols. You can use DECnet and TCP/IP simultaneously on an Ethernet and alternate between the two protocols on DDCMP point-to-point lines.

File access. Programs on any other DECnet Phase III/IV system can access DECnet–ULTRIX files for sequential reading, writing, directories, or deletion.

Access Control. DECnet–ULTRIX supports two ways for your client application to gain access to the server: access-control information and proxy.

See Sections 3.1.3 and 3.1.4 for instructions on how to use proxy and access-control information in a connection request.

1.2 Network Objects

In the DECnet–ULTRIX programming environment, a network object is a server application that can be accessed by name or number from other DECnet nodes. A client application identifies the server application it wants to connect to by specifying the server's object name or number as part of a connection request.

See the *DECnet–ULTRIX NCP Command Reference* manual for examples of network objects.

1.3 Communication Domains

A communication domain is a set of protocols that have common communication properties. DECnet–ULTRIX introduces the DECnet domain into the ULTRIX Interprocess Communication (IPC) environment for applications that communicate through the DECnet standard protocols.

1.4 Sockets

A socket is an addressable endpoint for communication. The client and server applications each create a socket that acts as a handle for sending and receiving data.

Each communication domain supports a different set of socket types. The DECnet communication domain supports the following socket types for DECnet–ULTRIX applications:

Sequenced-packet sockets. A sequenced-packet socket supplies a bidirectional, reliable, ordered, first-in,first-out (FIFO), unduplicated flow of data.

The socket preserves the record boundaries. A **write** operation transmits one message across the connection; a **read** operation—if it completes successfully—returns a single, logical message.

Stream sockets. A stream socket also supplies a bidirectional, reliable, ordered, FIFO, unduplicated flow of data.

Stream sockets provide a byte stream without using message boundaries. Data supplied as part of a **write** operation may or may not transmit a message across the connection, and a **read** operation may return data from one or more data packets. These possibilities depend on system variables unknown to the program.

1.5 Blocking and Nonblocking Input/Output Modes

Blocking and nonblocking are input/output (I/O) modes that cause a calling process to either wait (blocked) or not wait (nonblocked) for an I/O operation. Blocking prevents an I/O system call from returning control to a calling procedure until the operation completes. The nonblocking I/O mode returns control to the calling procedure immediately with an error message if there are not enough resources available to complete the operation.

1.6 Accept-Immediate and Accept-Deferred Modes

DECnet supports two modes for accepting incoming connections: immediate and deferred.

Accept-Immediate mode makes it possible for the server program to send and receive data as soon as the accept call operation completes. However, in this mode, the server does not have access to any optional data or access-control information that may have been supplied with the connection request.

Accept-Deferred mode lets the server program store, examine, and process any access-control information or optional data that is supplied as part of a connection request. The server must then accept or reject the connection.

As long as the socket is in accept-deferred mode, a server program can retrieve access-control information or retrieve and return optional data when a connect is pending; that is, after an **accept** call has successfully completed, but before the server accepts or rejects the connection.

1.7 Access-Control Information

The DECnet architecture lets the client requesting the connection pass access-control information to a server application. The server application then uses this information to determine if access should be granted. This information consists of three strings: *username, password,* and *account.* These are defined as follows:

username	A name of up to 39 characters assigned to the user on the server system.
password	A string of up to 39 characters that you use to gain access to the user account on the server system.
account	A string of up to 39 characters that some DECnet systems use to identify the remote users and their privileges upon logging in. The server ignores this string if it is not required.

Different servers interpret and use these strings according to their own requirements. In many cases, servers compare received access-control information against the system password file. Usually, the results of this comparison determine whether a connection request is accepted and, if it is, what privileges and quotas are allowed.

1.8 Proxy Access

In the DECnet domain, proxy access is a method of screening client application access to the server application without supplying a password.

When the client requests a connection, the node on which the client resides passes the identity of the client application to the target node on which the server resides. The supplied name (a log-in name or user ID) of the user initiating the request for the client must correspond with an entry listed in the target node's proxy access file.

This procedure is more secure than sending a password over the network.

1.9 Optional Data

In the DECnet domain, optional data is a string of up to 16 bytes that clients and servers can exchange on either a connect or disconnect sequence. This data is interpreted differently according to the application.

Some application protocols exchange additional identifying information (such as a protocol version number) at the time of the connection. This information is used to determine whether a connection request should be accepted. For example, DECnet Network Management uses optional data to exchange protocol version numbers before a connection is established. Also, when a socket is disconnected or a connection request is rejected, the application may use optional data to send an error message.

1.10 Out-of-Band Messages

An out-of-band message is an unsolicited, high-priority message that one application sends to another outside of the normal data channel. In most cases, it informs the receiving application of an unusual or abnormal event in the sending application.

Chapter 2

DECnet–ULTRIX Programming Tools

This chapter describes some of the tools for writing DECnet–ULTRIX client and server applications. It explains how the tools work and recommends methods of using them in the DECnet–ULTRIX programming environment.

Tools for programming client and server tasks in the DECnet domain include:

- The DECnet library (**libdnet.a**), which contains subroutines that simplify many basic programming operations. Two important subroutines included in this library are:
  - The **dnet_conn** subroutine, a routine that establishes a connection to a specified network object on a remote node.
  - The **dnet_eof** subroutine, a routine that tests the state of an established connection to a remote DECnet application.

NOTE

When you build programs that use these routines, you must specify **-ldnet** in the command line or makefile. Versions of the libraries suitable for use by **lint** are contained in the unsupported subset. See the *DECnet–ULTRIX Installation* manual for the subset name and location.

- The object spawner, a server program that listens for connection requests on behalf of all servers that are not actively listening for connection requests.
- System calls used within the DECnet domain to perform network connection and data transfer functions. Examples of these calls are **accept**, **bind**, **write**, and **close**.

2.1 How the dnet_conn Subroutine Works

When you use **dnet_conn** to establish a connection for a client application, the subroutine performs connection tasks in the following order:

1. Creates a socket in the DECnet domain.
2. Formats access-control information and optional connection data.
3. Issues a connection request to a server application.
4. Returns optional data received from the server if the connection is established.

The **dnet_conn** subroutine accepts the following as input:

- The name of the node to which you connect.

- The name or number of the server on the node.
- The socket type.
- A buffer for outgoing optional data, and a buffer for incoming optional data.

The **dnet_conn** subroutine also lets you pass access-control information to the server by appending it to the node name.

NOTE

The name of the node supplied to **dnet_conn** may be a node alias as defined in the **.nodes** file. Programs that use **dnet_conn** will prompt you for a password if you choose to omit the password field in an access-control string. To provide account security, the password that you type after the prompt does not echo.

If a connection is established successfully, **dnet_conn** returns a socket descriptor that can be used for subsequent **read** and **write** operations. If an error is encountered, **dnet_conn** returns a -1 value with additional error detail available in the external variable **errno**. Use the **nerror** system call to print out relevant DECnet error messages.

NOTE

The **dnet_conn** subroutine no longer returns the ULTRIX diagnostic message [ECONNABORTED] or [ECONNRESET]. If DECnet is not installed on the system, the **socket** request that **dnet_conn** makes will fail with the ULTRIX error message [EPROTONOSUPPORT], which is equivalent to the DECnet error message, "Protocol not supported."

2.2 How the dnet_eof Subroutine Works

When you use **dnet_eof** to test the state of an established connection, the subroutine performs the following steps:

1. Tests a DECnet socket to determine if an end-of-file (EOF) condition exists.
2. Returns a value of 0 if it determines a connection is in an active state.
3. Returns a nonzero value if it determines that a connection is in an inactive state.

This subroutine is useful for determining if any data exists for a **read** operation and if the socket is connected.

See **dnet_eof(3dn)** for more information about how to use the **dnet_eof** subroutine.

2.3 How the DECnet Object Spawner Works

The logic sequence for using the object spawner follows:

1. The object spawner creates a socket in the DECnet domain and listens for incoming connection requests on behalf of multiple DECnet objects (named and numbered).
2. When the object spawner receives a request to connect to an object, the spawner checks the object database to verify that either the server program or the **default** object is defined. If neither is found, it rejects the connection.

3. If access-control information is specified with the connection request, the object spawner verifies the information with the system password file. If the information is invalid, the connection is rejected. If it is valid, go to step 7.
4. If proxy is requested with the connection request, the object spawner verifies the proxy request with the system proxy file. If no entry is found, the object spawner uses the default user associated with the object entry for the server.
5. If access-control information is not specified or proxy is not requested, the object spawner uses the default user associated with the object entry for the server.
6. If the default user is specified for the object entry and defined in the system password file, the connection is accepted. Otherwise, the connection is rejected.
7. The object spawner redirects standard input and standard output to the network connection.
8. The object spawner executes the server program after setting up the environment. The environment is based on information in the password file entry of the user through which access was granted to the object. In addition, the setting for the process group is set equal to the process ID, and the group access list is initialized. Standard error is then redirected to **/dev/null**.

The logic sequence for server programs using the object spawner depends on the specified mode of acceptance, as follows:

1. If the accept mode is immediate, the object spawner completes the server connection. The server program can then exchange data by reading standard input and writing standard output.
2. If deferred mode was chosen, the connection request must be completed by the server using either standard input or standard output as the socket handle. The server program uses the **getsockopt** and **setsockopt** calls to complete the connection, as shown in steps 6 through 8 of the system call logic sequence described in Section 2.4.1.

2.4 How DECnet–ULTRIX System Calls Work

The following sections describe how client and server applications use system calls to establish a connection, exchange data, and terminate a connection.

2.4.1 Using System Calls for Client Programs

The following list shows how a client application can use system calls to establish a network connection:

1. To initiate a connection, a client program creates a socket for the connection by issuing a **socket** call. This call creates a socket of the specified type in the DECnet domain. The **socket** call returns a descriptor for the socket, which is used for subsequent program requests.
2. To set up access control, use one or both of the following methods:
   a. To pass access-control information, issue the **setsockopt** call on the descriptor returned from the socket call with the option DSO_CONACCESS. The structure **accessdata_dn**, which contains the data you have specified, is passed as a parameter to the call.

b. To use proxy access, set the SDF_PROXY bit in the *sdn_flags* field of the **sockaddr_dn** structure before issuing the **connect** call. If the program is to be executed with superuser privileges, you may want to bind a name to be used as the proxy source name to the socket, before issuing the **connect** call.

3. To pass optional data, issue **setsockopt** again with the option DSO_CONDATA. The structure **optdata_dn**, which contains the data you have specified, is passed as a parameter.
4. A client program issues a **connect** call to request a connection to a specified object. If the preceding **setsockopt** calls were successful, DECnet will use the data you have supplied when the **connect** call is issued.
5. The client program can issue a **getsockopt** call to retrieve incoming optional data.
6. If the **connect** call is successful, the program can use the socket to send and receive data by means of the **read** and **write** or **recv** and **send** calls.
7. The program can use the **getsockopt** or **setsockopt** call to send or receive optional data with the **close** call.
8. The **close** call terminates the connection.

Table 2-1 summarizes the logic sequences for a typical DECnet–ULTRIX client application using system calls. See Appendix B for programming examples.

Table 2–1: Client Program Calling Sequences

Function	System Calls
Create a socket for the connection.	**socket**
Send optional data and/or access-control information with the connection request.	**setsockopt***
Define a name for the socket.	**bind***
Request a connection to a server program.	**connect**
Retrieve optional data from the server.	**getsockopt** *
Transfer normal data.	**send** **recv** **read** **write**
Transfer out-of-band data.	**send** * **recv** *
Send or receive optional data with the **close** call	**setsockopt*** **getsockopt** *
Terminate the connection.	**close**

*Optional.

2.4.2 Using System Calls for Server Programs

The following list describes the logic sequence for a typical DECnet–ULTRIX server program using system calls to establish a connection.

1. The server program creates a socket in the DECnet domain by issuing a **socket** call.

2. The object name or number is stored in the **sockaddr_dn** structure. The server issues a **bind** call to assign the object name or number. (See the description of the **bind** call in **bind(2dn)**).
3. A server can issue a **setsockopt** call to set the mode of acceptance to deferred.
4. A server issues a **listen** call, which declares that the socket is available for receiving connection requests directed to the bound name.
5. An **accept** call completes when the system receives a connection request. (Note that an **accept** call in deferred mode must be issued to receive a request, but does not actually accept the connection.) If the **accept** is successful, a new server socket is created.

NOTE

If you specified accept-immediate mode, you can use the socket to send and receive data. If you specified accept-deferred mode, however, you must complete the following steps before attempting to transfer data.

6. A server issues a **getsockopt** call to retrieve any access-control information or optional data that was supplied with the **connect** call.
7. A server issues a **setsockopt** call to supply any optional data that it wants to return to the client program.
8. The server issues a **setsockopt** call to accept or reject the connection.
9. If the server accepts the connection, the program can use the socket to send and receive data by means of the **read** and **write** or **recv** and **send** calls.
10. The program can use the **getsockopt** or **setsockopt** call to send or receive optional data with the **close** call.
11. The **close** call terminates the connection.

Table 2-2 summarizes the calling sequences for a typical DECnet–ULTRIX server application using system calls. See Appendix B for programming examples.

Table 2–2: Server Program Calling Sequences

Function	System Calls
Create a socket to listen for connection requests.	**socket**
Define a name for the socket.	**bind**
Set the mode of acceptance. The default mode is IMMEDIATE, and the other possible mode is DEFERRED.	**setsockopt** *
Declare the socket available for connection requests.	**listen**
Block the server program until it receives a connection request.	**accept**
When in accept-deferred mode, receive optional data or access-control information.	**getsockopt** *
When in accept-deferred mode, supply optional data, such as the server software version number.	**setsockopt** *

*Optional

(continued on next page)

Table 2–2 (Cont.): Server Program Calling Sequences

Function	System Calls
When in accept-deferred mode, accept or reject the connection.	setsockopt *
Transfer normal data.	send recv read write
Transfer out-of-band data.	send* recv *
Send or receive optional data with the close call.	setsockopt * getsockopt *
Terminate the connection.	close

*Optional

2.5 Three Ways to Use DECnet–ULTRIX Programming Tools

You can use DECnet–ULTRIX tools to establish a connection between client and server applications in three ways:

- Let DECnet–ULTRIX subroutines and the DECnet object spawner handle programming tasks for you.
- Use system calls to perform the same tasks yourself.
- Combine DECnet–ULTRIX subroutines and DECnet object spawner functions with system call functions.

After you have established a connection between the client and server, you can use system calls and subroutines to perform data transfer tasks and disconnect the network connection.

2.5.1 Using dnet_conn with the DECnet Object Spawner

You can let the **dnet_conn** subroutine and the object spawner establish the connection between the client and server. **dnet_conn** initiates the connection and performs connection tasks for the client application. The object spawner performs connection tasks on behalf of the server.

The object spawner is the recommended tool for server programs because it provides the following services:

- Eliminates the need for coding connection request processing, including access-control information and proxy handling, in the server program.
- Reduces the number of idle processes because it listens on behalf of multiple servers.

See Appendix B for a programming example that shows how you can use **dnet_conn** and the object spawner to establish a connection between client and server applications.

2.5.2 Using DECnet–ULTRIX System Calls

You can use DECnet–ULTRIX system calls to establish a session and accept connection requests. The system calls can perform all the tasks that **dnet_conn** performs for you. They can also give you more programming flexibility by letting you control each task during the connection process.

See Appendix B for programming examples.

2.5.3 Combining DECnet–ULTRIX Tools

You can combine tools to establish a session. For example, you can use **dnet_conn** to initiate a connection request for the client and use the system calls to accept the request for the server. Also, you can use system calls to initiate a connection request for the client and let the object spawner accept the request for the server. Table 2-3 shows four ways to establish a session with DECnet–ULTRIX tools.

Table 2–3: Methods for Establishing a Client–Server Session

Client Application Task	Server Application Task
dnet_conn requests a session.	DECnet object spawner establishes the connection.
dnet_conn requests a session.	System calls accept or reject the request.
System calls request a session.	DECnet object spawner accepts the request.
System calls request a session.	System calls accept or reject the request.

Chapter 3

Programming in the DECnet Domain

This chapter explains procedures for writing DECnet–ULTRIX applications by using either subroutines or system calls. The following sections explain how to perform four basic network application programming tasks:

- Program a client application to initiate a network connection.
- Program a server application to respond to a network connection request.
- Perform data transfer tasks after establishing a connection.
- Disconnect a session between a client and server application.

3.1 How to Program a Client-Initiated Connection

To initiate a network connection between your client application and a remote DECnet application, you can use **dnet_conn** or system calls to:

1. Choose the socket type for the client.
2. Identify the node and server to which the client is attempting to connect.
3. Set up either access-control or proxy for client access to servers.
4. Send and receive optional data.

3.1.1 Choosing a Socket Type for the Client

DECnet supports two socket types: sequenced-packet sockets and stream sockets. Table 3-1 compares these sockets.

Table 3–1: Socket Types Compared

Sequenced-Packet Socket	Stream Socket
Preserves message boundaries	Does not preserve message boundaries
DECnet-VAX default socket	Commonly used for ULTRIX applications
Not available on TCP/IP	Available on TCP/IP

When writing applications, use the same socket type as the program you connect to uses. If a connection uses both types of sockets, the program using the stream socket:

- Has no control over how much data the sequenced-packet socket will get when it performs its next **read** operation.

- Does not receive any indication of the record boundary on a **read** operation–even though the program using the sequenced-packet socket creates a record boundary with each **write** operation.

Applications that use message boundaries to interpret data cannot be used in connections using both stream and sequenced-packet sockets. If you cannot guarantee that both ends of a connection will use the same socket type, design an application protocol that does not use record boundaries.

3.1.1.1 Using dnet_conn

To specify the socket type as an argument in **dnet_conn**, use this format:

s=dnet_conn(*node,object,type,*)

where

type is either SOCK_STREAM, if you want to specify a stream socket, or SOCK_SEQPACKET, if you want to specify a sequenced-packet socket. If the socket type is set to 0, use the default, SOCK_SEQPACKET.

EXAMPLE:

This example shows a sequenced-packet socket being selected for node NAVAHO with object 17.

```
s=dnet_conn(NAVAHO,17,SOCK_SEQPACKET,)
```

For more information about socket types, see the description in **dnet_conn(2dn)**.

3.1.1.2 Using System Calls

To specify a socket type during a **socket** call operation, use this format:
s=socket(*node,object,type,*)

where

type is either SOCK_STREAM, if you want to indicate a stream socket, or SOCK_SEQPACKET, if you want to indicate a sequenced-packet socket.

EXAMPLE:

This example shows a sequenced-packet socket being selected for a DECnet node.

```
s=socket(AF_DECnet,SOCK_SEQPACKET,)
```

For more information about socket types, see the description in **socket(2dn)**.

3.1.2 Specifying a Node and Server

Before your client application can request a connection, you must:

1. Identify the node on which the server resides.

   You can use a node name (an alphanumeric string of one to six characters) or node address (an area number from 1 to 63, followed by a period and a node number from 1 to 1,023).

2. Identify the server you want to connect to.

   The client application can specify the server application in one of two ways:

   - By a network object name of up to 16 characters.
   - By a network object number from 1 to 255.

If the remote object is defined (that is, an object number has been assigned to the server process—either by Digital or by the user), the object number is the recommended method for requesting the network service to avoid any conflicts in naming conventions.

See the *DECnet–ULTRIX NCP Command Reference* manual for detailed information on preassigned network object numbers and procedures for defining network objects in the object database.

3.1.2.1 Using dnet_conn

To specify a node and server while requesting a connection, use the following format:
dnet_conn(*node,object,*)

where

node	is either the node name or address used to specify the node.
object	is either the object name or the object number used to specify the server.

The following examples show you how to use **dnet_conn** to specify a node and a server by name and number while establishing a connection.

EXAMPLE 1:

In this example, the client program uses **dnet_conn** to connect to object number 17 on node NAVAHO.

```
s=dnet_conn("navaho","#17",)
```

EXAMPLE 2:

In this example, the client program uses **dnet_conn** to connect to object **xyz** on the node with a DECnet address 55.342.

```
s=dnet_conn("55.342","xyz",)
```

3.1.2.2 Using System Calls

To specify the node and server, you must use the **sockaddr_dn** data structure.

EXAMPLE 1:

In this example, the client program connects to server **xyz** on node 55.342:

```
#define SERVERNAME "xyz" ❶
#define NODE       "55.342"
        .
        .
        .
struct sockaddr_dn sockaddr; ❷
struct dn_naddr*node_addr;
int sock;
        .
        .
        .
bzero(&sockaddr, sizeof(sockaddr));

sockaddr.sdn_family = AF_DECnet; ❸

sockaddr.sdn_objnamel = strlen(SERVERNAME); ❹
    strncpy(sockaddr.sdn_objname,SERVERNAME,
sizeof(sockaddr.sdn_objname));

node_addr = dnet_addr(NODE); ❺

sockaddr.sdn_add = *node_addr;

if (connect(sock, &sockaddr, sizeof(sockaddr)) < 0)
{

      perror("connect");
      exit(1);

}
        .
        .
        .
```

COMMENTS:

❶ Defines the server using a network object name. This name can be up to 16 characters long.

❷ The **sockaddr_dn** data structure is defined in the file **/sys/netdnet/dn.h** (see Appendix A).

❸ You must specify the AF_DECnet address family.

❹ These three lines show how the server name is put into the **sockaddr_dn** structure.

❺ If you are using the **dnet_addr** subroutine to specify a node in the **sockaddr_dn** data structure, you must use a node address.

EXAMPLE 2:

This example shows you how to connect to server 17 on node NAVAHO by number.

```
#define SERVERNUMBER 17 ❶
#define NODENAME     "navaho"
                .
                .
                .
struct sockaddr_dn sockaddr;  ❷
struct nodeent *nodep;
int sock;
                .
                .
                .
bzero(&sockaddr, sizeof(sockaddr));

sockaddr.sdn_family = AF_DECnet;  ❸
sockaddr.sdn_objnum = SERVERNUMBER;

nodep = getnodebyname(NODENAME);  ❹
bcopy(nodep->n_addr, sockaddr.sdn_nodeaddr, nodep->n_length);  ❺
sockaddr.sdn_nodeaddrl = nodep->n_length;

if (connect(sock, &sockaddr, sizeof(sockaddr)) < 0)
{
        perror("connect");
        exit(1);
}
                .
                .
                .
```

COMMENTS:

❶ Defines the server using a network object number. This can be any number from 1 to 255.

❷ The **sockaddr_dn** data structure is defined in the file **/sys/netdnet/dn.h** (see Appendix A).

❸ You must specify the AF_DECnet address family.

❹ If you are using the **getnodebyname** subroutine to specify a node in the **sockaddr_dn** structure, you must specify a node name.

❺ These three lines show how to fill in the node address fields of the **sockaddr_dn** data structure.

See Appendix B for more detailed programming examples.

3.1.3 Specifying Access-Control Information

You can use either the **dnet_conn** subroutine or system calls to specify access-control information for the client application.

3.1.3.1 Using dnet_conn

To specify access-control information, use the following format:
dnet_conn(*"node/username[/password][/account]"*)

where

node	is a string that specifies the node name or address, followed by the *username*, *password*, and *account* strings separated by slashes (/).
username	is a name of up to 39 characters assigned to the user on the server system.
password	is a string of up to 39 characters that you use to gain access to the user account on the server system.
account	is a string of up to 39 characters used by some DECnet systems. The server ignores this string if it is not required.

3.1.3.2 Using System Calls

To specify access-control information, issue a **setsockopt** call with the DSO_CONACCESS option. The access-control data is passed in the **accessdata_dn** data structure (described in Appendix A). The access-control information is used when the client issues the **connect** call.

EXAMPLE:

This example shows a **setsockopt** call being issued with a DSO_CONACCESS option in the **accessdata_dn** structure.

```
set_access_control(socket, user, password) ❶
int socket;
char *user, *password;

{
    struct accessdata_dn acc_data;

    bzero(&acc_data, sizeof(acc_data));

    acc_data.acc_userl = strlen(user);
    strncpy(acc_data.acc_user, user, acc_data.acc_userl);

    acc_data.acc_passl = strlen(password);                    ❷
    strncpy(acc_data.acc_pass, password, acc_data.acc_passl);

    return(setsockopt(sock, DNPROTO_NSP, DSO_CONACCESS,
           &acc_data, sizeof(acc_data)));
}
```

COMMENTS:

❶ The lengths of the user name and password are defined in **/sys/netdnet/dn.h**.

❷ The *account* string is not used in this example.

NOTE

The **setsockopt** call must precede the **connect** call to supply access information for the connection request.

After the client issues a **connect** call, DECnet flushes any access-control information previously set with the **setsockopt** call and the DSO_CONACCESS option. Therefore, you must specify new access-control data for any subsequent connection requests that the client issues on the same socket.

3.1.4 Requesting Proxy

You can use either the **dnet_conn** subroutine or system calls to request proxy for a client application.

3.1.4.1 Using dnet_conn

The **dnet_conn** subroutine requests proxy by default. If the default (proxy) setting is not changed, **dnet_conn** binds the user's log-in name (converted to uppercase) to the socket. This bound name is used as the source name for the outgoing connection only when a program's user ID is set to root or invoked by the superuser. Otherwise, the ASCII form of the user's ID is used as the source name for proxy access.

If you do not want **dnet_conn** to request proxy access at the remote system, set the external variable, *proxy_requested*, equal to zero.

EXAMPLE:

In this example, the *proxy_requested* variable is set to zero.

```
extern char proxy_requested;
proxy requested=0;
```

3.1.4.2 Using System Calls

To request proxy, set the SDF_PROXY bit in the *sdn_flags* field of the **sockaddr_dn** structure before issuing the **connect** call.

EXAMPLE:

In this example, the client program issues a request for proxy access.

```
#define SERVERNAME "xyz"
#define NODE       "55.342"
     .
     .
     .
     struct sockaddr_dn sockaddr;
     struct sockaddr_dn bindaddr;
     struct dn_naddr *node_addr;
     char *user_name, *getlogin();
     int sock, len, status;
     .
     .
     .
     bzero(&sockaddr, sizeof(sockaddr));
     bzero(&bindaddr, sizeof(bindaddr));

     if (((user_name = getlogin()) == NULL) || (*user_name == NULL))
         user_name = "anonymous";

     bindaddr.sdn_family = AF_DECnet;
     len = strlen(user_name);
     if (len > sizeof(bindaddr.sdn_objname))
         len = sizeof(bindaddr.sdn_objname); ❶
     bindaddr.sdn_objnamel = len;
     strncpy(bindaddr.sdn_objname, user_name, len);
```

```
if (bind(sock, &bindaddr, sizeof(bindaddr)))
{
    perror("bind");
    exit(1);
}
sockaddr.sdn_flags |= SDF_PROXY; ❷
sockaddr.sdn_family = AF_DECnet;
sockaddr.sdn_objnamel = strlen(SERVERNAME);
strcpy(sockaddr.sdn_objname, SERVERNAME);

node_addr = dnet_addr(NODE);

sockaddr.sdn_add = *node_addr;
.
.
.
status = connect(sock, &sockaddr, sizeof(sockaddr));
.
.
.
```

COMMENTS:

❶ Bind a name to the socket before issuing the **connect** call.

❷ If your program is running with superuser privileges, the name you bind to the socket is used as the source name for proxy access.

NOTE

If you do not bind a name to the socket, or if you issue a **connect** call without root privileges, DECnet–ULTRIX uses the user's ID in ASCII as the source name for proxy access.

3.1.5 Setting Up Optional Data for the Client

Use **dnet_conn** or system calls to exchange up to 16 bytes of optional data while a connection is being established.

3.1.5.1 Using dnet_conn

The **dnet_conn** subroutine lets you specify a buffer containing optional data to be sent to the server. It also lets you specify a buffer containing any optional data returned by the server. After a client program sends optional data with a connection request, DECnet flushes the data.
To specify optional data, use the following format:

dnet_conn (**node,object,type,***opt_out,opt_outl,opt_in,opt_inl*)

where

opt_out	specifies the address of the outgoing data.
opt_outl	specifies the length of the outgoing data.
opt_in	specifies the address of the buffer that will store the optional data returned by the server.
opt_inl	is the address of an integer that specifies the size of that buffer before the call and will contain the actual number of bytes of optional data returned by the server on successful completion of **dnet_conn**.

EXAMPLE 1:

You can specify no outgoing optional data to be supplied and no incoming optional data to be expected. For example, when using a sequenced-packet socket to connect to object SOAPBOX on node ALEXUS, issue the following call:

```
s=dnet_conn("alexus","soapbox", SOCK_SEQPACKET,0,0,0,0);
```

EXAMPLE 2:

In this example, the client program connects to the network management object **nml** (object number 19).

```
char in_data[16], out_data[] = {4,0,0}; ❶
int in_length, out_length = sizeof(out_data);
int sock;
    .
    .
    .
    sock = dnet_conn("alexus/root", ❷
                     "#19", 0, ❸
                     out_data, out_length, ❹
                     in_data, &in_length); ❺
    .
    .
    .
```

COMMENTS:

❶ This example shows the current version of the NICE protocol used by **nml**.

❷ This call would be issued if you wanted to connect to node ALEXUS as user ROOT.

❸ In this example, a socket type of 0 defaults to SOCK_SEQPACKET.

❹ The protocol version number is sent as optional data and a version number is expected in return.

❺ A buffer is provided in which the **nml** object can return its protocol version number.

3.1.5.2 Using System Calls

To specify that optional data is to be sent to the server, use the **optdata_dn** structure. To set up the connection data to send to the server, use the **setsockopt** call.

NOTE

You must set the optional data each time you reissue the connection request.

EXAMPLE 1:

In this example, the client program sends three bytes of optional data to the server.

```
char version[] = {4, 0, 0};
        .
        .
        .
        struct optdata_dn out_opt; ❶
        int sock;
        .
        .
        .
        bzero(&out_opt, sizeof(out_opt));
```

```
out_opt.opt_optl = sizeof(version);                    ❷
bcopy(version, out_opt.opt_data, out_opt.opt_optl);

if (setsockopt(sock, DNPROTO_NSP, DSO_CONDATA,
               &out_opt, sizeof(out_opt)) < 0)
{
        perror("setsockopt");
        exit(1);
}
.
.
.
```

COMMENTS:

❶ Defined in the **/sys/netdnet/dn.h** file.

❷ Optional data is copied into the **optdata_dn** structure. The data length is also put into this structure.

After the connection has been established, use the **getsockopt** call to retrieve the data returned by the server program.

EXAMPLE 2:

In this example, up to 16 bytes of data are placed in *in_opt* and the data count is placed in *in_opt_len*.

```
struct optdata_dn in_opt;
int sock, in_opt_len;
.
.
.
bzero(&in_opt, sizeof(in_opt));

in_opt_len = sizeof(in_opt);

if (getsockopt(sock, DNPROTO_NSP, DSO_CONDATA,
               &in_opt, &in_opt_len) < 0)
{
        perror("getsockopt");
        exit(1);
}
.
.
.
```

3.2 How to Establish a Connection for the Server

The DECnet domain supports two methods for programming the server:

- Use the DECnet object spawner to listen for connection requests on behalf of your server. When the spawner receives a request for your server, it executes a copy of your server and redirects standard input and standard output to the network connection.
- Use system calls to set up a server that can run independently as a daemon.

You can use either method to perform the following tasks:

- Specify the socket type: stream or sequenced-packet.
- Assign a name to the server.
- Select an accept mode: immediate or deferred.
- Verify remote user access to the server.

- Exchange optional data with client applications.

3.2.1 Choosing a Socket Type for the Server

When writing a server application, you must choose either of the two available socket types: sequenced-packet or stream. (Table 3-1 compares the characteristics of each socket type.)

NOTE

Be sure to use the same socket type as for the client application.

3.2.1.1 Using the Network Control Program (NCP)

The DECnet object spawner uses the object database, which consists of entries defined by the network manager. One of the characteristics defined for the object entry is the socket type. Use **ncp** commands to choose between a stream socket and sequenced-packet socket. For example, to define a stream socket as the socket type for an object entry, you can use the **ncp** command **set object**:

set object *object-name type*

where

object *object-name*	Specifies that parameters are to be created or modified for the named object only (a maximum of 16 alphanumeric characters).
type	is either SOCK_STREAM, if you are specifying a stream socket, or SOCK_SEQPACKET, if you are specifying a sequenced-packet socket.

EXAMPLE:

This example shows how to use the **set object** command to choose a socket type for object entry **myserver**.

```
>ncp RET
ncp>set object myserver type stream RET
```

See the *DECnet–ULTRIX NCP Command Reference* manual for more information about using **ncp** commands.

3.2.1.2 Using System Calls

To specify a socket type, use the following format:

s=socket (*node,object,type,*)

where

type	is either SOCK_STREAM, if you are specifying a stream socket, or SOCK_SEQPACKET, if you are specifying a sequenced-packet socket.

EXAMPLE:

This example shows how to use the **socket** call to specify a socket type.

```
s=socket (AF_DECnet,SOCK_STREAM,0,);
```

3.2.2 Assigning a Name to Your Server

All server applications must have an object name or number associated with them. Object names contain 1 to 16 alphanumeric characters. Object numbers range from 0 to 255; however, some numbers are reserved for certain types of applications. Table 3-2 shows these assignments:

Table 3–2: Object Number Assignments

Object Number	Type of Application
1-127	Reserved for DECnet-supplied programs, such as **fal** and **dlogin**.
128-255	Should be used if you are writing a server application that performs a known network service.

3.2.2.1 Using the Object Spawner

If you have chosen an object number from 1 to 255, you must define your object name and number in the object database.
EXAMPLE:

In the following example, the **ncp set object** command defines the Network Management Listener (**nml**) as object number 19.

```
% ncp RET
ncp>set object nml number 19 RET
```

(For more details, see the *DECnet–ULTRIX Network Management* manual.)

If your object number is 0, you can define it in the database or the spawner will use the entry for the **default** object.

NOTE

The **default** DECnet object is no longer shipped with a default user defined for it. Therefore, incoming connections to this object without valid access-control information or proxy will not be accepted. If you want to allow unrestricted access to your system through this object, issue the following **ncp** command:

```
% ncp define object default user guest RET
```

In previous DECnet–ULTRIX releases, the process group ID for processes created by the DECnet spawner was set to 0. Starting with Version 2.2, the process group ID setting is equal to the process ID setting.

3.2.2.2 Using System Calls

You must specify your object name, object number, or both in a **sockaddr_dn** structure. If you are using object number 0, you must also specify an object name. You can then issue a **bind** call on the same socket you use to listen for calls.
EXAMPLE:

In this example, 128 is specified as an object number.

```
int s;
struct sockaddr_dn server;

server.sdn_family = AF_DECnet;
server.sdn_objnum = 128;
if (bind(s,&server, sizeof(struct sockaddr_dn))<0)

    exit();
```

3.2.3 Selecting Accept-Immediate or Accept-Deferred Modes

The server uses accept-immediate or accept-deferred mode to accept incoming connections. (See **accept(2dn)** for details on how to use the **accept** call to establish a network connection.)

NOTE

Accept-immediate mode is the default setting for both the DECnet object spawner and system calls.

3.2.3.1 Using the Object Spawner

Use the **set object** command to choose between accept-immediate and accept-deferred modes for an object entry.

EXAMPLE:

This example shows you how to use the **set object** command to select accept-deferred mode for the server, **myserver**.

```
% ncp RET
ncp>set object myserver accept deferred RET
```

For more information about **ncp** commands, see the *DECnet–ULTRIX NCP Command Reference* manual.

3.2.3.2 Using System Calls

Use the **setsockopt** call to select accept-immediate or accept-deferred mode. Specify the ACC_IMMED option to select accept-immediate mode; specify the ACC_DEFER option to select accept-deferred mode.

EXAMPLE:

In this example, the socket accept mode is set to deferred.

```
char val = ACC_DEFER;

setsockopt(s,DNPROTO_NSP,DSO_ACCEPTMODE,&val,sizeof(val));
```

3.2.4 Verifying Remote User Access to the Server

A server can screen incoming connection requests from the client based on the access-control or proxy information the client supplies.

3.2.4.1 Using the Object Spawner

Regardless of the accept mode chosen for the server, the spawner verifies access-control information or proxy information that the client supplies. The spawner also rejects or processes connection requests based on the following conditions:

- If the information is invalid, the spawner rejects the connection request.
- If the information is valid, the spawner processes the **connect** request according to the accept mode specified for the server in the object database.
- If immediate mode was specified, the spawner accepts the request and initiates the server.
- If deferred mode was specified, the spawner initiates the server. The server must then use the **setsockopt** call with either the DSO_CONACCEPT or DSO_CONREJECT option to either accept or reject the request.

See Section 2.3 for more information about how the object spawner uses access-control or proxy information.

3.2.4.2 Using System Calls

If you are not using the spawner to process requests for the server, you can use the **getsockopt** system call to screen requests. If the server is bound and it is listening on a specific address for its own connection requests, it can verify access to the service based on incoming access-control information or proxy.

EXAMPLE:

In this example, the **getsockopt** call retrieves incoming access-control information.

```
struct accessdata_dn acc_data;

getsockopt(s,DNPROTO_NSP,DSO_CONACCESS,&acc_data,sizeof(acc_data));
```

NOTE

A process has access to data in the password field of the **accessdata_dn** structure only if it is running with superuser privileges. If not, the password field of the **accessdata_dn** structure will be null.

3.2.5 Exchanging Optional Data

In the DECnet domain, client and server can exchange up to 16 bytes of optional data during the connection and disconnection processes. Both server and client interpret this data according to an application-specific design. The following steps describe how the client and server applications exchange optional data:

1. Before the server can read optional connection data, you must ensure that the socket is in accept-deferred mode.
2. If the server is going to accept the connection, use the **setsockopt** call with the DSO_CONDATA option to specify the outgoing optional data.
3. To accept the connection, issue the **setsockopt** call with the DSO_CONACCEPT option.
4. If the server is going to reject the connection, specify outgoing optional data using the **setsockopt** call with the DSO_DISDATA option.

5. To reject the connection, issue the **setsockopt** call with the DSO_CONREJECT option.

EXAMPLE 1:

This example shows how the **optdata_dn** structure specifies optional data before accepting or rejecting a connection.

```
struct optdata_dn optional;
char message[] = { 1, 2, 3, 4, 5 };

bzero(&optional, sizeof(optional));
bcopy(message, optional.opt_data, sizeof(message));
optional.opt_optl = sizeof(message);
```

EXAMPLE 2:

In this example, the server program sends optional data and accepts a connection.

```
setsockopt(sock, DNPROTO_NSP, DSO_CONDATA, &optional,
           sizeof(optional));
setsockopt(sock, DNPROTO_NSP, DSO_CONACCEPT, 0, 0);
```

EXAMPLE 3:

In this example, the server program sends optional data and rejects a connection.

```
setsockopt(sock, DNPROTO_NSP, DSO_DISDATA, &optional,
           sizeof(optional));
setsockopt(sock, DNPROTO_NSP, DSO_CONREJECT, 0, 0);
```

3.3 Transferring Data After Establishing a Connection

After establishing a connection, the client and server applications use their sockets to send and receive data via the **send**, **recv**, **write**, and **read** system calls.

NOTE

If you are using the DECnet object spawner, standard input and standard output are redirected to the network connection.

DECnet–ULTRIX software supports the following services during data transfer between client and server applications:

- Blocking or nonblocking input/output modes
- Out-of-band messages
- Zero-length message detection on sequenced packet sockets

3.3.1 Using Blocking or Nonblocking I/O

Blocking mode is the default mode for ULTRIX software. Unless otherwise specified, an ULTRIX **read** call blocks a calling process until data is available for the **read** operation, and an ULTRIX **write** call blocks a calling process until enough resources are available to buffer data for the **write** operation.

If no data is available when a **read** call is issued, or if there are not enough resources to buffer data for a **write** operation, a nonblocking I/O call returns control to the calling process immediately with an EWOULDBLOCK message. Otherwise, the calling procedure regains control as soon as the **read** or **write** operation completes.

EXAMPLE:

To set up the nonblocking I/O mode, include the ULTRIX **fcntl(2)** system call in the beginning of your application, as follows:

```
fcntl(sock,F_SETFL,FNDELAY);
```

3.3.2 Using Out-of-Band Messages

An application can send an out-of-band message from 1 to 16 bytes long ahead of normal data messages. However, it can send only one out-of-band message over a socket at a time.

The receiving application must read any pending out-of-band message before the sending program can send another. The signal SIGURG indicates the arrival of out-of-band data.

To send or receive an out-of-band message, an application must specify the MSG_OOB flag with the **send** or **recv** call, depending on the following conditions:

- The **send** call specifies the socket used to send an out-of-band message and the buffer used to contain the message.
- The **recv** call specifies the socket used to receive an out-of-band message and the buffer used to contain the message.

EXAMPLE 1:

In this example, the application sends "buffer" as an out-of-band message:

```
char buffer[] = { 1, 2, 3, 4, 5 };
send(sock, buffer, sizeof(buffer), MSG_OOB);
```

You can also use the **select** call to wait for out-of-band data. When the **select** call returns and indicates that an out-of-band message is present, you can use a **recv** call to read the message.

EXAMPLE 2:

In this example, the application uses the **select** call to determine if out-of-band data has arrived.

```
int EMask;
EMask = 1<<sock;
select(sock+1, (int *)0, (int *)0, &EMask, (struct timeval *)0);

if( EMask & 1<<sock )
    msgsize = recv(sock, buffer, sizeof(buffer), MSG_OOB);
```

3.3.3 Detecting Zero-Length Messages

On a sequenced-packet socket, a returned value of zero on a **read** operation indicates that either the end of the file has been reached or a zero-length message has been received. An end-of-file status on a socket indicates that the logical link was disconnected and communication over the socket is not possible.

To distinguish between an end-of-file message and a zero-length packet, use the **dnet_eof** subroutine. If the return value from the **dnet_eof** call is zero, a zero-length packet has been received. Otherwise, the logical link has been disconnected.

EXAMPLE:

This example shows how to use **dnet_eof** to distinguish zero-length messages from end-of-file messages.

```
/* Read the next packet on a DECnet sequenced packet socket */
length = read(sock, buff, buffsize);
if( length == -1 )
/* read failed, refer to read(2dn) for more information */
else if( length == 0  &&  dnet_eof(sock) )
/* End Of File has been reached */

/* If here, then we have successfully read a packet */
```

3.4 How to Disconnect a DECnet Connection

Either the client or the server application can initiate the disconnection of a DECnet connection. The application that initiates the disconnection can also specify the optional disconnect data to send to the other application.

To disconnect a DECnet connection:

1. Before initiating the disconnection, the application can specify between 1 and 16 bytes of optional disconnection data by issuing a **setsockopt** call with a DSO_DISDATA option.
2. The application either closes all references to the DECnet socket or issues a single **shutdown** call to request disabling of the **send** or **send** and **recv** operations. The **shutdown** call is used when the application is set to reference the DECnet socket after the connection is terminated.
3. If an application does not initiate the disconnection, it can retrieve the optional disconnection data sent by the application that initiated the disconnection. The application can issue a **getsockopt** call with the DSO_DISDATA option.

EXAMPLE 1:

In this example, the application terminates the DECnet connection while sending optional disconnection data.

```
struct optdata_dn disdata;
char message[] = {1, 2, 3, 4, 5};

/* Prepare optional disconnect data */
bzero(&disdata, sizeof(disdata));
bcopy(message, disdata.opt_data, sizeof(message));
disdata.opt_optl = sizeof(message);

setsockopt(sock,DNPROTO_NSP,DSO_DISDATA,&disdata,sizeof(disdata))S ();
close(sock);
```

EXAMPLE 2:

In this example, the application determines that the DECnet connection has been terminated, and retrieves any optional data that may have been sent by the application that terminated the connection.

```
struct optdata_dn disdata;
int length;

length = read(sock, buff, buffsize);

/* Check to see if the connection has been disconnected */
if(length == 0  &&  dnet_eof(sock))
{

int structsize;
```

```
    /* Retrieve any optional disconnect data that was sent */
    structsize = sizeof(disdata);
    getsockopt(sock,DNPROTO_NSP,DSO_DISDATA,&disdata,&structsize);

/* No longer need the socket descriptor. Closing it will free
up local network resources that were associated with it */
close(sock);

    }
```

NOTE

The successful completion of a **write** call does not necessarily indicate that the data has already been sent to the remote node. A successful **write** operation means that the local system has accepted the data and will transmit it as soon as possible.

The effect of issuing a **close** call while data that has not been sent is queued for a remote application depends on the value of the LINGER option, which is set through the **setsockopt** call. If the SO_LINGER option is set, the **close** operation will be delayed while an attempt is made to send or acknowledge all data. Otherwise, the data in the queue is eliminated and the system processes the **close** operation with an abort condition.

Part II

Reference

Chapter 4

DECnet–ULTRIX System Calls

This reference chapter describes DECnet–ULTRIX system calls in detail. The format for this information corresponds to that in the ULTRIX reference pages. See the *ULTRIX Reference* manuals for more information about format.

Each system call begins a separate page in alphabetical order. The name of the system call appears in a running head followed by the appropriate section number and a suffix. For example, **accept(2dn)** appears on the reference pages describing the **accept** call. The **2** indicates that the section describes system calls. The **dn** indicates that the system call is used in the DECnet domain.

4.1 System Call Summary

Table 4-1 summarizes the function of each DECnet–ULTRIX system call.

Table 4–1: DECnet–ULTRIX System Calls

System Call	Function
accept	Accepts a connection request.
bind	Binds a name to a socket.
close	Terminates a logical link and deactivates a socket descriptor.
connect	Initiates a connection request.
getpeername	Returns the name of the peer connected to a socket.
getsockname	Returns the current name of a socket.
getsockopt	Returns the options associated with a socket.
listen	Listens for pending connection requests.
read	Reads (receives) data.
recv	Receives normal data and out-of-band messages.
select	Performs synchronous I/O multiplexing.
send	Sends data and out-of-band messages.
setsockopt	Sets socket options.
shutdown	Shuts down a DECnet connection.
socket	Creates a new socket.
write	Writes (sends) data.

4.2 On-Line Manual Pages

The system call descriptions also appear as on-line documentation in **accept(2dn)**, **bind(2dn)**, **close(2dn)**, and so on.

4.3 Format and Conventions

The descriptions of the DECnet–ULTRIX system calls have the following format:

SYNTAX

Gives the complete syntax for the system call. The following conventions apply to syntax lines:

command	Indicates terms that are constant and must be typed exactly as presented.
...	Indicates that the preceding item can be repeated one or more times.
italics	Indicate a variable, for which either you or the system must specify a value.
%	The default user prompt in multiuser mode.
#	The default superuser prompt.

DESCRIPTION

Supplies function and background information.

RETURN VALUE

Explains the meaning of a value returned by a utility when it completes or does not complete an operation.

DIAGNOSTICS

Lists diagnostic messages that can be returned.

RESTRICTIONS

Describes restrictions that apply to the use of the system call or subroutine.

SEE ALSO

Provides cross-references to associated information in this manual and in other DECnet–ULTRIX and ULTRIX manuals.

In text, cross-references to specific manual reference pages include the section number in the ULTRIX or DECnet-ULTRIX reference manual where the commands are documented. For example, socket(2dn) refers to the description of the socket system call in Section 2dn of the ULTRIX reference pages.

4.4 System Call Descriptions

The following pages describe each system call in detail.

accept (2dn)

NAME

accept — accept a connection request

SYNTAX

```
#include <sys/types.h>\bold
#include <sys/socket.h>\bold
#include <netdnet/dn.h>\bold

ns = accept (s,addr,addrlen)
int ns,s;
struct  sockaddr_dn *addr;
int *addrlen;
```

where

Input Arguments:

s — specifies a descriptor for a socket that has been returned by the **socket** call, bound to a name by the **bind** call, and is listening for connection requests after issuing a **listen** call.

addr — is the address of a **sockaddr_dn** structure. This address identifies the source entity that is requesting the connection.

addrlen — specifies the size of the source address.

Return Arguments:

ns — is the new descriptor for the accepted socket.

addr — specifies the address of a **sockaddr_dn** structure. This call fills in the following data fields:

- *sdn_family* — specifies the communications domain as AF_DECnet.
- *sdn_objnum* — specifies the source DECnet object number.
- *sdn_objnamel* — is the size of the source node's object name. This argument is used only when the DECnet object number, *sdn_objnum*, is 0.
- *sdn_objname* — defines the name of the network program, which can be up to 16 characters. This argument is used only when the DECnet object number, *sdn_objnum*, is 0.
- *sdn_nodeaddrl* — is the size of the node address for the source program.
- *sdn_nodeaddr* — specifies the node address for the source program.

addrlen — specifies the actual length (in bytes) of the returned address.

DESCRIPTION

The **accept** call extracts the first connection request on the queue of pending connections, creates a new socket having the same properties as *s*, and allocates a new file descriptor, *ns*, for the socket. *s* remains open and listens for connection requests.

There are two modes of accepting an incoming connection: immediate and deferred. You can specify the mode of acceptance with the **setsockopt** call.

Accept-immediate mode, specified as ACC_IMMED, causes both client and server programs to complete the protocol exchange at the Network Services Protocol (NSP) level. The server program ignores any access-control information or optional data that the source program may have sent. ACC_IMMED is the default.

Deferred mode, specified as ACC_DEFER, causes the **accept** call to be completed by a server program without a full protocol exchange between itself and the client program. Deferred mode allows a server program to examine and process any access-control information or optional data before notifying the client program of the acceptance or rejection of its connect request.

RETURN VALUE

If the **accept** call succeeds, it returns a nonnegative integer, which is a descriptor for the accepted socket. If an error occurs, the call returns a value of -1 and the external variable **errno** will contain the type of error.)

DIAGNOSTICS

The call succeeds unless:

[EBADF]	The *s* argument is not a valid descriptor.
[ECONNABORTED]	DECnet is shutting down on the local node.
[EFAULT]	The *addr* argument is not in a write-enable part of the user address space.
[EWOULDBLOCK]	The socket is marked for nonblocking, and no connections are waiting to be accepted.

SEE ALSO

bind(2dn), listen(2dn), setsockopt(2dn), socket(2dn)

bind (2dn)

NAME

bind — bind a name to a socket

SYNTAX

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdnet/dn.h>

bind (s,name,namelen)

int s;
struct sockaddr_dn name;
int namelen;
```

where

s specifies a descriptor for a socket that has been returned by the **socket** call.

name specifies the address of a **sockaddr_dn** structure.

namelen specifies the size of the address of the **sockaddr_dn** structure. This call fills in the following data fields:

sdn_family	specifies the communications domain as AF_DECnet.
sdn_objnum	specifies the DECnet object number to which you are binding. If the object number is 0, the DECnet object name, *sdn_objname*, is used.
sdn_objnamel	specifies the size of the object name to which you are binding. This argument is used only when the DECnet object number, *sdn_objnum*, is 0.
sdn_objname	specifies the name of the source network program. This argument is used only when the DECnet object number, *sdn_objnum*, is 0.

DESCRIPTION

The **bind** call assigns a name to a socket. When a socket is created with the **socket** call, it exists in a name space (address family) but has no assigned name. The **bind** call requests that a specific name be assigned to the socket.

NOTE

The DECnet object numbers 1 through 127 are reserved by Digital for sockets that are in programs running as superuser.

RETURN VALUE

If the **bind** call is successful, a value of 0 is returned. If the call is unsuccessful, a value of -1 is returned and the external variable **errno** contains error detail.

DIAGNOSTICS

The call succeeds unless:

[EACCESS]	The requested address is reserved, and the current user does not have superuser privileges.
[EADDRINUSE]	The specified name is already bound to a listening socket on the local machine.
[EAFNOSUPPORT]	The *sdn_family* is not AF_DECnet.
[EBADF]	The *s* argument is not a valid descriptor.
[EFAULT]	The *name* argument is not located in a valid part of the user's address space.
[EINVAL]	The *namelen* argument is not the size of *sockaddr_dn*, or *sdn_objnamel* is not in the range of 0 to 16.

RESTRICTION

Bound names are ignored for sockets set up to initiate connections in all programs except programs running as superuser.

SEE ALSO

socket (2dn)

close (2dn)

NAME

close — terminate a DECnet connection

SYNTAX

```
close (s)
int s;
```

where

s specifies a descriptor for a socket that has been returned by the **socket** or the **accept** call.

DESCRIPTION

The **close** call terminates an outstanding connection over a DECnet socket descriptor and deactivates the descriptor. When the last **close** call is issued on that descriptor, all associated naming information and queued data are discarded. (The **close** call deletes a descriptor from the per-process object reference table. If this is the last reference to the underlying socket, the socket is deactivated.)

The effect of issuing a **close** call while unsent data is queued for a remote program depends on the value of the linger option set with the **setsockopt** call. If SO_LINGER is set, the **close** operation is delayed while an attempt is made to send or acknowledge all data; otherwise, the data in the queue is eliminated and the system processes the **close** with an abort.

A **close** of all of a program's descriptors is automatic when an **exit** call is issued, but because there is a limit on the number of active descriptors per program, the **close** call is necessary for programs that deal with many descriptors.

RETURN VALUE

If the call succeeds, a value of 0 is returned. If an error occurs, a value of -1 is returned. Additional error detail is specified in the external variable **errno**.

DIAGNOSTICS

The call succeeds unless:

[EBADF] The *s* argument is not a valid descriptor.

SEE ALSO

accept(2dn), setsockopt(2dn), socket(2dn)

connect (2dn)

NAME

connect — initiate a connection request

SYNTAX

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdnet/dn.h>
connect (s,name,namelen)
int s;
struct  sockaddr_dn * name;
int namelen;
```

where

s — specifies a descriptor for a socket that has been returned by the **socket** call.

name — specifies the address of a **sockaddr_dn** structure. This address identifies the destination process for the **connect**.

namelen — specifies the size of the address of the **sockaddr_dn** structure. This call fills in the following data fields:

- *sdn_family* — specifies the communications domain as AF_DECnet.
- *sdn_flags* — specifies the object flag option, which you can use to request outgoing proxy.
- *sdn_objnum* — specifies the DECnet object number to which you are connecting. If the object number is 0, the DECnet object name, *sdn_objname*, is used.
- *sdn_objnamel* — specifies the size of the object name to which you are connecting. This argument is used only when the DECnet object number, *sdn_objnum*, is 0.
- *sdn_objname* — specifies the name of the destination program. This argument is used only when the DECnet object number, *sdn_objnum*, is 0.

DESCRIPTION

The **connect** call issues a connect request to a server program. The server program is specified by the *name* argument, which is an address in the DECnet domain.

Optional user data and access-control information are passed with the **connect** call if this data is previously set with the **setsockopt** call.

RETURN VALUE

If the **connect** succeeds, a value of 0 is returned. If the **connect** fails, a value of -1 is returned and the external variable **errno** contains error detail.)

DIAGNOSTICS

The call succeeds unless:

[EACCESS]	The access-control information was rejected.
[EADDRNOTAVAIL]	No such node.
[EAFNOSUPPORT]	Addresses in the specified address family cannot be used with this particular socket.
[EBADF]	The *s* argument is not a valid descriptor.
[ECONNREFUSED]	The attempt to connect was refused by the remote object.
[EFAULT]	The *name* argument specifies an area outside the process address space.
[EHOSTDOWN]	Local node is shut down.
[EHOSTUNREACH]	Node unreachable.
[EINVAL]	The *namelen* argument is not the size of **sockaddr_dn**, or *sdn_objnamel* is not in the range of 0 to 16.
[EISCONN]	The socket is already connected.
[ENETDOWN]	The remote node is shutting down.
[ENOSPC]	There are no resources available for a new connection at either the local or remote system.
[ESRCH]	Unrecognized object at the remote node.
[ETIMEDOUT]	Connection establishment was timed out before a connection was established.
[ETOOMANYREFS]	The remote object has too many active connections.
[INPROGRESS]	The socket is nonblocking and the connection is in progress. To determine when the connection has either completed or failed, select the socket for **write** using the **select** call.

SEE ALSO

accept(2dn), close(2dn), setsockopt(2dn), socket(2dn)

dnet_conn(3dn)

getpeername (2dn)

NAME

getpeername — get name of connected peer

SYNTAX

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdnet/dn.h>
getpeername (s,name,namelen)
int s;
struct  sockaddr_dn *name;
int *namelen;
```

where

Input Arguments:

s — specifies a descriptor for a socket that has been returned by the **socket** or the **accept** call.

name — specifies the address of a **sockaddr_dn** structure.

namelen — specifies the length of the address of the **sockaddr_dn** structure.

Return Arguments:

name — specifies the address of a *sockaddr_dn* structure. This call fills in the following data fields:

- *sdn_family* — specifies the communications domain as AF_DECnet.
- *sdn_objnum* — specifies the DECnet object number for the peer program. If the object number is 0, the DECnet object name, *sdn_objname*, is used.
- *sdn_objnamel* — is the length of the peer object name. This argument is used only when the object number, *sdn_objnum*, is 0.
- *sdn_objname* — specifies the name of the peer network program, which can be up to a 16-element array of characters. This argument is used only when the object number, *sdn_objnum*, is 0.
- *sdn_nodeaddrl* — is the size of the remote node address.
- *sdn_nodeaddr* — specifies the remote node address.

namelen — returns the length of the address of a **sockaddr_dn** structure.

DESCRIPTION

The **getpeername** call returns the name of the peer DECnet program connected to a specified socket.

RETURN VALUE

If the call succeeds, a value of 0 is returned. If an error occurs, a value of -1 is returned. When an error condition exists, the external variable **errno** contains error detail.

DIAGNOSTICS

The call succeeds unless:

[EBADF]	The *s* argument is not a valid descriptor.
[EFAULT]	The *name* argument points to memory located in an invalid part of the process address space.
[ENOBUFS]	Insufficient resources were available in the system to perform the operation.
[ENOTCONN]	The socket *s* is not connected.

getsockname (2dn)

NAME

getsockname — return the current name for a socket

SYNTAX

```
getsockname (s,name,namelen)
int s;
struct  sockaddr_dn *name;
int *namelen;
```

where

Input Arguments:

s	specifies a descriptor for a socket that has been returned by the **socket** or the **accept** call.
name	specifies the address of a **sockaddr_dn** structure. This address is the name that was bound to the socket.
namelen	specifies the length of the address of the **sockaddr_dn** structure.

Return Arguments:

name	specifies the address of a **sockaddr_dn** structure. This call fills in the following data fields:
sdn_family	specifies the communications domain as AF_DECnet.
sdn_objnum	specifies the DECnet object number for the socket. If the object number is 0, the DECnet object name, *sdn_objname*, is used.
sdn_objnamel	is the size of the object name. This argument is used only when the object number, *sdn_objnum*, is 0.
sdn_objname	specifies the DECnet object name, which can be up to a 16-element array of characters. This argument is used only when the object number, *sdn_objnum*, is 0.
sdn_nodeaddrl	is the size of the local node address.
sdn_nodeaddr	specifies the local node address.
namelen	returns the length of the address of the **sockaddr_dn** structure.

DESCRIPTION

The **getsockname** call returns the current name of a specified socket.

RETURN VALUE

If the call succeeds, a value of 0 is returned. If an error occurs, a value of -1 is returned. When an error condition exists, the external variable **errno** contains error detail.

DIAGNOSTICS

The call succeeds unless:

[EBADF]	The *s* argument is not a valid descriptor.
[EFAULT]	The *name* argument points to memory located in an invalid part of the process address space.
[ENOBUFS]	Insufficient resources were available in the system to perform the operation.

SEE ALSO

bind(2dn)

getsockopt (2dn) and setsockopt (2dn)

NAME

getsockopt, setsockopt — get and set options on sockets

SYNTAX

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdnet/dn.h>

setsockopt (s,level,optname,optval,optlen)
int s,level,optname;
char *optval;
int *optlen;

getsockopt (s,level,optname,optval,optlen)
int s,level,optname;
char *optval;
int *optlen;
```

where

Input Arguments:

s — specifies a descriptor for a socket in the AF_DECnet domain.

level — specifies the level at which options are interpreted:

The **socket** call level SOL_SOCKET causes options to be interpreted at the socket level.

A level of DNPROTO_NSP instructs DECnet to interpret an option.

optname — specifies an option to be interpreted at the level specified.

optval, optlen — specify option values that are used with the **getsockopt** and **setsockopt** calls. The interpretations of these arguments relative to each call are as follows:

getsockopt:

optval — specifies the buffer that will contain the returned value for the requested option.

optlen — is the value result parameter that initially contains the size of the buffer pointed to by *optval* and is modified on return to indicate the actual length of the returned value.

setsockopt:

optval — specifies the address for the buffer that contains information for setting option values.

optlen — specifies the length of the option value buffer.

DESCRIPTION

The **getsockopt** and **setsockopt** calls manipulate various options associated with a socket. Options may exist at multiple levels, so you must specify the level for the desired operation. The socket level SOL_SOCKET options are as follows:

SO_DEBUG — enables the recording of debugging information.

SO_LINGER — controls the actions taken when unsent messages are queued on a socket and a **close** call is issued. If SO_LINGER is set, the system will block the process until all data has been received by the remote system or until it is unable to deliver the information.

At the DECnet level, socket options can define the way in which a pending connection is accepted. Options at this level also control the sending and receiving of optional user data and access-control information, and return information on current link status. The DECnet options follow:

DSO_ACCEPTMODE — The accept mode option is used at the DECnet level for processing **accept** calls. The program must issue a **bind** call on the socket before this option is valid. A **setsockopt** call to set the accept mode is valid only if issued before a **listen** call is performed.

There are two values that can be supplied for this option: ACC_IMMED (immediate mode) and ACC_DEFER (deferred mode). (The optional value (*optval*) for this option is the *char* type.)

ACC_IMMED — The default mode for this option. When immediate mode is in effect, control is immediately returned to a server program following an **accept** call with the connection request accepted.

ACC_DEFER — Enables a server program to complete an **accept** call without fully completing the connection to the client program. The server program can then examine the access-control information and/or optional data before accepting or rejecting the pending connection. The server program can then issue the **setsockopt** call with the appropriate reject or accept option.

DSO_CONACCEPT — Allows the server program to accept the connection on socket returned by the **accept** call and previously set to ACC_DEFER mode. Any optional data previously set by DSO_CONDATA will be sent. (There is no optional value (*optval*) for this option.)

DSO_CONREJECT — Lets the server program reject a pending connection on a socket returned by the **accept** call and previously set to ACC_DEFER mode. Any optional data previously set by DSO_DISDATA will be sent. The reject reason is passed with this option as a *short int* value.

DSO_CONDATA

This option allows up to 16 bytes of optional user data to be sent by the **setsockopt** call. The data is sent as a result of the **connect** or the **accept** (with the deferred option) call. The optional data is passed in an **optdata_dn** structure. (See the **optdata_dn** data structure in Appendix A for formatting information.) The data is read by the task issuing the **getsockopt** call with this option.

DSO_DISDATA

Allows up to 16 bytes of optional data associated with the **setsockopt** call. It is sent as a result of the **close** call. The optional data is passed in a **optdata_dn** structure. (See the **optdata_dn** data structure in Appendix A for formatting information.) The data is read by the program issuing the **getsockopt** call with this option.

DSO_CONACCESS

Allows access-control information to be set by the client program and received by the server program. The data is sent with the **setsockopt** call and passed to the server when an ensuing **connect** call is issued. The server retrieves the information by issuing a **getsockopt** call. The access data is sent to the server program. It is passed with the **connect** call in an **accessdata_dn** data structure. (See the **accessdata_dn** data structure in Appendix A for formatting information.) The access data is read by the program issuing the **getsockopt** call with this option.

NOTE

The DSO_CONACCESS socket option for the **getsockopt** call will function only in programs running as root.

DSO_LINKINFO

Gets information on the state of a DECnet logical link. Link state information is passed in a *linkinfo_dn* structure in the *idn_linkstate* field. The following are possible link states and their respective values:

```
LL_INACTIVE          logical link inactive
LL_CONNECTING        logical link connecting
LL_RUNNING           logical link running
LL_DISCONNECTING     logical link disconnecting
```

See Appendix A for information on formatting for the **linkinfo_dn** data structure.

RETURN VALUE

If the call completes successfully, a value of 0 is returned. If the call fails, a value of -1 is returned and the external variable **errno** contains error details.

DIAGNOSTICS

The call succeeds unless:

[EBADF]	The *s* argument is not a valid descriptor.
[EBUSY]	The pending connection has gone away.
[EDOM]	The acceptance mode is not valid.
[EFAULT]	The options are not located in a valid part of the process address space.
[EMSGSIZE]	The size of the option buffer is incorrect.
[ENOBUFS]	No buffer space is available to return access-control data.
[ENOPROTOOPT]	No access-control information was supplied with the connection request.
[EOPNOTSUPP]	The option is unknown.

listen (2dn)

NAME

listen — listen for pending connect requests

SYNTAX

```
listen (s,backlog)
int s,backlog;
```

where

s — specifies a descriptor for a socket that has been returned by the **socket** call and bound to a name by the **bind** call.

backlog — defines the maximum length for the queue of pending connection requests for a particular socket. If a connection request arrives when the queue is full, the connection will be rejected.

DESCRIPTION

The **listen** call declares a socket as being available to receive connection requests and listens for incoming connections. The **listen** call must be issued before the server program accepts an incoming connection request.

RETURN VALUE

If the call succeeds, a value of 0 is returned. If an error occurs, a value of -1 is returned. When an error condition exists, the external variable **errno** contains error details.

DIAGNOSTICS

The call succeeds unless:

[EADDRINUSE] — The name bound to the socket is already being used for a listen socket.

[EBADF] — The *s* argument is not a valid descriptor.

SEE ALSO

accept(2dn), connect(2dn), socket(2dn)

read (2dn)

NAME

read — read or receive data

SYNTAX

```
#include <sys/types.h>
#include <sys/sockets.h>

cc = read (s,buf,buflen)
int cc,s;
char *buf;
int buflen;
```

where

Input Arguments:

s	specifies a descriptor for a socket that has been returned by the **socket** call.
buf	specifies the address of buffer into which data is read.
buflen	specifies the size of the message buffer.

Return Arguments:

cc	is the length of the returned message.

DESCRIPTION

The **read** call is used to read normal data messages from another DECnet program. You can use **read** only on a connected socket. If no messages are available at the socket, the **read** call waits for a message to arrive. However, if the socket is nonblocking, a status of -1 is returned with the external variable **errno** set to EWOULDBLOCK.

You can use the **select** call to determine when more data will arrive.

The length of the message is returned in *cc*. If a message is too long to fit in the supplied buffer, the excess bytes can be discarded, depending on the type of socket from which the message is received. Sequenced-packet sockets discard extra bytes. Stream sockets store extra bytes in the kernel and use them for the next **read** call.

RETURN VALUE

If the call succeeds, the number of bytes actually read and placed in the buffer are returned. The system reads the number of bytes requested only if the descriptor references a file containing that many bytes before the end of file. If the end of file has been reached, a value of 0 is returned.

NOTE

A returned value of 0 can also indicate that a zero-length message has been received on a sequenced-packet socket. See **dnet_eof(3dn)** for more information.

If an error occurs, a value of -1 is returned and the global variable **errno** is set to indicate the error.

DIAGNOSTICS

The call succeeds unless:

[EBADF]	The *s* argument is not a valid file descriptor open for reading.
[EFAULT]	The *buf* argument points outside the allocated address space.
[EINTR]	A **read** operation from a slow device was interrupted by the delivery of a signal before any data arrived.
[EWOULDBLOCK]	The socket is marked nonblocking and the **read** operation would have blocked.

SEE ALSO

connect(2dn), socket(2dn)

dnet_eof(3dn)

dup(2), socketpair(2)

recv (2dn)

NAME

recv — receive normal data and out-of-band messages

SYNTAX

```
#include <sys/types.h>
#include <sys/socket.h>

cc = recv (s,buf,buflen,flags)
int cc,s;
char *buf;
int buflen,flags;
```

where

Input Arguments:

s	specifies a descriptor for a socket that has been returned by the **socket** call.
buf	specifies the address of the buffer that will contain the received message.
buflen	specifies the size of the message buffer.
flags	are set to MSG_PEEK, which looks at incoming messages, or to MSG_OOB, which indicates that a program will receive out-of-band messages.

Return Arguments:

cc	is the length of the returned message.

DESCRIPTION

The **recv** call is used to receive normal or out-of-band data from another DECnet program. **recv** can be used only on a connected socket (see **connect(2dn)**). If no messages are available at a socket, the **recv** call waits for a message to arrive. However, if the socket is nonblocking, a status of -1 is returned with the external variable **errno** set to EWOULDBLOCK.

Use the **recv** call instead of the **read** call when you want to specify the MSG_PEEK and MSG_OOB *flags* arguments to look at incoming messages and to receive out-of-band messages.

You can use the **select** call to determine when more data will arrive.

The length of the message is returned in *cc*. If a message is too long to fit in the supplied buffer, the excess bytes may be discarded depending on the type of socket from which the message is received. Sequenced-packet sockets discard extra bytes. Stream sockets store extra bytes in the kernel and use them for the next **recv** call.

The *flags* argument for a **recv** call is formed by **oring** one or more of the following values:

```
#define    MSG_PEEK  0x1  /* peek at incoming message */
#define    MSG_OOB   0x2  /* process out-of-band data */
```

Out-of-band messages are sent to a receiving program ahead of normal data messages. Out-of-band messages are sent and received as DECnet interrupt messages and can be from 1 to 16 bytes in length. The signal SIGURG indicates the arrival of out-of-band data. You can also use the **select** call to determine if out-of-band data has arrived by using the *exceptfds* argument.

RETURN VALUE

If the call succeeds, the number of received characters is returned. If an error occurs, a value of -1 is returned. Additional error detail will be specified in the external variable **errno**.

DIAGNOSTICS

The call succeeds unless:

[EBADF]	The *s* argument is not a valid descriptor.
[EFAULT]	The data was specified to be received into a nonexistent or protected part of the process address space.
[EWOULDBLOCK]	The socket is marked nonblocking and the **receive** operation would have blocked.

RESTRICTION

The MSG_PEEK *flags* argument cannot be used with out-of-band messages.

SEE ALSO

read(2dn), send(2dn), socket(2dn), write(2dn)

select (2dn)

NAME

select — synchronous I/O multiplexing

SYNTAX

```
#include <sys/time.h>

nfound = select (nfds,readfds,writefds,exceptfds,timeout)
int nfound,nfds, *readfds, *writefds, *exceptfds;
struct timeval *timeout;
```

where

Input Arguments:

nfds	specifies the number of descriptors to be checked. For example, the bits from 0 to *nfds*-#1 in the masks are examined.
readfds	specifies the descriptor to be examined for read (or receive) data ready. This descriptor can be given as a null pointer (0) if it is of no interest.
writefds	specifies the descriptor to be examined for write (or send) data ready. This descriptor can be passed as a null pointer (0) if it is of no interest.
exceptfds	specifies the descriptor to be examined for out-of-band data ready. This descriptor can be given as a null pointer (0) if it is of no interest.
timeout	specifies an address of a **timeval** structure. If *timeout* is a nonzero pointer, it specifies a maximum interval to wait for the selection to complete. If *timeout* is a zero pointer, the select blocks indefinitely. To affect a poll, *timeout* should be nonzero, pointing to a zero-valued *timeval* structure.

Return Argument:

nfound	is the total number of ready descriptors returned.

DESCRIPTION

The **select** call examines the I/O descriptors specified by the bit masks *readfds*, *writefds*, and *exceptfds* to determine whether the descriptors are ready for reading or writing or have an out-of-band condition pending, respectively. File descriptor *f* is represented by the bit 1<<f in the mask. The *nfds* descriptors are checked; that is, the bits from 0 through *nfds* -#1 in the masks are examined. The **select** call returns a mask of those descriptors that are ready. The total number of ready descriptors is returned in *nfound*.

If *timeout* is a nonzero pointer, it specifies a maximum interval to wait for the selection to complete. If *timeout* is a zero pointer, the **select** call blocks indefinitely. To affect a poll, the *timeout* argument should be nonzero and pointing to a zero-valued **timeval** structure.

The *readfds*, *writefds*, and *exceptfds* arguments can be defined as 0 if no descriptors are of interest.

RETURN VALUE

The **select** call returns the number of descriptors that are contained in the bit masks. If an error occurs, a value of -1 is returned. Additional error details will be contained in the external variable **errno**. If the time limit expires, a value of 0 is returned.

DIAGNOSTICS

The call succeeds unless:

[EBADF]	One of the bit masks is specified as an invalid descriptor.
[EINTR]	An asynchronous signal was delivered before any of the selected events occurred or the time limit expired.

RESTRICTION

The descriptor masks are always modified on return, even if the call returns as the result of the timeout.

SEE ALSO

accept(2dn), connect(2dn), read(2dn), recv(2dn), send(2dn), write(2dn)

send (2dn)

NAME

send — send normal data and out-of-band messages

SYNTAX

```
#include <sys/types.h>
#include <sys/socket.h>

cc = send (s,msg,msglen,flags)
int cc,s;
char *msg;
int msglen,flags;
```

where

Input Arguments:

s	specifies a descriptor for a socket that has been returned by the **socket** call.
msg	specifies the address of the buffer that contains the outgoing message.
msglen	specifies the size of the message.
flags	are set to MSG_OOB, which sends an out-of-band message.

Return Argument:

cc	is the number of characters sent.

DESCRIPTION

The **send** call transmits normal or out-of-band data to another program. It can be used only when a socket is in a connected state. See **connect(2dn)** for more information.

Use the **send** call instead of **write** when you want to specify the MSG_OOB *flags* argument to indicate that out-of-band data will be sent to the destination program. Out-of-band messages are sent to a receiving program ahead of normal data messages. Out-of-band messages are sent and received as DECnet NSP interrupt messages and can be from 1 to 16 bytes long.

The number of characters sent is returned in *cc*. If no message space is available at the receiving socket to hold the message being transmitted, the **send** call will, in most cases, block. If the socket is set in nonblocking I/O mode, **send** returns an error with **errno** set to EWOULDBLOCK.

You can use the **select** call to determine when it is possible to send more data.

RETURN VALUE

If the call succeeds, the number of characters sent are returned. If an error occurs, a value of -1 is returned. Additional error detail will be specified in the external variable **errno**.

DIAGNOSTICS

The call succeeds unless:

[EBADF]	The *s* argument is not a valid descriptor.
[EFAULT]	An invalid user address space was specified for an argument.
[EMSGSIZE]	The socket requires that the message be sent atomically, but the size of the message made this impossible. Note that zero-length messages are illegal.
[EWOULDBLOCK]	The socket is marked nonblocking and the **send** operation would have blocked.

SEE ALSO

read(2dn), recv(2dn), write(2dn)

setsockopt (2dn)

NAME

See **getsockopt(2dn)**

shutdown (2dn)

NAME

shutdown — shut down a logical link

SYNTAX

```
shutdown (s,how)
int s,how;
```

where

s	is a descriptor for the socket associated with the DECnet logical link that you want to shut down.
how	is an integer specifying how the connection is shut down. If the value of *how* is 0, further receives are disabled. If the value of *how* is 1, further sends are disabled. If the value of *how* is 2, further sends and receives are disabled.

DESCRIPTION

The **shutdown** call shuts down all or part of a DECnet logical link connection on the socket specified by the argument *s*.

RETURN VALUE

If the call succeeds, a value of 0 is returned. If the call fails, a value of -1 is returned.

DIAGNOSTICS

<he call succeeds unless:

[EBADF]	The *s* argument is not a valid descriptor.
[ENOTCONN]	The specified socket is not connected.
[ENOTSOCK]	The *s* argument is a file, not a socket.

SEE ALSO

connect(2dn), socket(2dn)

socket (2dn)

NAME

socket — create a socket and return a descriptor

SYNTAX

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdnet/dn.h>

s = socket (af,type,protocol)
int s,af,type,protocol;
```

where

Input Arguments:

af specifies the address format for the DECnet communication domain as AF_DECnet.

type specifies the socket type. The DECnet domain supports the following socket types:

- SOCK_STREAM. Stream sockets provide bidirectional, reliable, sequenced, and unduplicated byte streams.
- SOCK_SEQPACKET. Sequenced-packet sockets provide bidirectional, reliable, sequenced data flow while preserving record boundaries in data.

protocol specifies the protocol to be used with the socket. Valid protocols are 0 (default) and DNPROTO_NSP (DECnet protocol). If you specify the socket type SOCK_SEQPACKET, you must set the protocol to zero.

Return Argument:

s is the value for the socket descriptor.

DESCRIPTION

The **socket** call creates a socket and returns a socket descriptor.

A socket is an addressable endpoint of communication. A program uses the socket to transmit and receive data to and from a similar socket in another program. Subsequent calls on a particular socket reference that socket's descriptor.

RETURN VALUE

If the call completes successfully, a socket descriptor value is returned. This descriptor is used for subsequent system calls on this particular socket. If an error occurs, a value of -1 is returned. Additional error detail is contained in the external variable **errno**.

DIAGNOSTICS

The call succeeds unless:

[EAFNOSUPPORT]	The specified address family is not supported in this version of the system.
[EMFILE]	Too many open files.
[ENFILE]	The per-process descriptor table is full.
[ENOBUFS]	No buffer space is available. The socket cannot be created.
[EPROTONOSUPPORT]	The specified protocol is not supported.
[ESOCKTNOSUPPORT]	The specified socket type is not supported in this address family.

SEE ALSO

dnet_conn(3dn)

write (2dn)

NAME

write — write or send data

SYNTAX

```
#include <sys/types.h>
#include <sys/socket.h>

cc = write (s,msg,msglen)
int cc,s;
char *msg;
int msglen;
```

where

Input Arguments:

s	specifies a descriptor for a socket that has been returned by the **socket** call.
msg	specifies the address of the buffer that contains the outgoing message.
msglen	specifies the size of the message.

Return Argument:

cc	is the number of bytes sent.

DESCRIPTION

The **write** call is used to transmit normal data messages to another program. You can use **write** only when a socket is in a connected state. See **connect(2dn)** for more information.

The number of bytes sent is returned in *cc*. If no message space is available at the receiving socket to hold the message being transmitted, the **write** call will, in most cases, block. If the socket is set in nonblocking I/O mode, the **write** returns in error with **errno** set to EWOULDBLOCK.

You can use the **select** call to determine when it is possible to send more data.

RETURN VALUE

If the call succeeds, the number of bytes actually written is returned. If an error occurs, a value of -1 is returned and **errno** is set to indicate the error.

DIAGNOSTICS

The call succeeds unless:

[EBADF]	The *s* argument is not a valid descriptor open for writing.
[EFAULT]	Part of *s* or data to be written to the file points outside the process's allocated address space.
[EMSGSIZE]	An attempt is made to transmit a zero-length message or a message that is larger than the DECnet pipeline quota on a sequenced-packet socket.
[EPIPE]	An attempt is made to write to a socket type SOCK_STREAM, which is not connected to a peer socket.
[EWOULDBLOCK]	The socket is marked nonblocking and the **write** operation would have blocked.

SEE ALSO

connect(2dn), read(2dn), recv(2dn), send(2dn)

Chapter 5

DECnet–ULTRIX Subroutines

This reference chapter describes the DECnet–ULTRIX subroutines, which you can find in the C library, **/lib/libdnet.a**. The format for this information corresponds to that in the ULTRIX reference pages. See the *ULTRIX Reference* manuals for more information about format.

Each subroutine begins a separate page in alphabetical order. The name of the subroutine appears in a running head followed by the appropriate section number and a suffix. For example, **dnet_addr(3dn)** appears on the pages describing the **dnet_addr** subroutine. The **3** indicates that the section describes subroutines. The **dn** indicates that the subroutine is used in the DECnet domain.

5.1 Subroutine Summary

Table 5-1 summarizes the function of each DECnet–ULTRIX subroutine.

Table 5–1: DECnet–ULTRIX Subroutines

Subroutine	Function
dnet_addr	Converts an ASCII node address to binary.
dnet_conn	Connects to a specified network object on a remote node and sends any access-control information or optional user data.
dnet_eof	Tests the current state of a connection to determine whether the end-of-file has been reached.
dnet_getallas	Gets extended node information.
dnet_htoa	Returns a DECnet ASCII node name that corresponds to a 16-bit binary node address contained in a structure of the type **dn_naddr**. If a node name is not found for the node address, **dnet_htoa** returns an ASCII string representation of the node address.
dnet_ntoa	Converts a 16-bit binary node address, which is contained in a structure of the type **dn_naddr**, to its ASCII string representation.
dnet_otoa	Converts a DECnet object name or number to its ASCII string representation.

(continued on next page)

Table 5–1 (Cont.): DECnet–ULTRIX Subroutines

Subroutine	Function
getnodeadd	Returns a pointer to a structure of the type **dn_naddr**, which contains your local DECnet–ULTRIX node address.
getnodeent	Accesses the network node database and returns node information.
getnodename	Returns an ASCII string representation of your local DECnet–ULTRIX node name.
nerror	Produces DECnet error messages.

5.2 On-Line Manual Pages

The subroutine descriptions appear also as on-line documentation; for example, **dnet_addr(3dn)**, **dnet_conn(3dn)**, **dnet_eof(3dn)**, and so on.

See the *DECnet–ULTRIX Use* manual for instructions on how to use on-line manual pages.

5.3 Format and Conventions

The descriptions of the DECnet–ULTRIX system calls have the following format:

SYNTAX

Gives the complete syntax for the subroutine. Syntax lines use the graphic conventions described at the end of this chapter. The following conventions apply to syntax lines:

`command`	Indicates terms that are constant and must be typed exactly as presented.
...	Indicates that the preceding item can be repeated one or more times.
italics	Indicate a variable, for which either you or the system must specify a value.
%	The default user prompt in multiuser mode.
#	The default superuser prompt.

DESCRIPTION

Supplies function and background information.

RETURN VALUE

Explains the meaning of a value returned by a subroutine when it completes or does not complete an operation.

DIAGNOSTICS

Lists diagnostic messages that can be returned.

RESTRICTIONS

Describes restrictions that apply to the use of the subroutine.

SEE ALSO

Provides cross-references to associated information in this manual and in other DECnet–ULTRIX and ULTRIX manuals.

In text, cross-references to specific manual reference pages include the section number in the ULTRIX or DECnet-ULTRIX reference manual where the commands are documented. For example, dnet_conn(3dn) refers to the description of the dnet_conn subroutine in Section 3dn of the ULTRIX reference pages.

5.4 Subroutine Descriptions

The following pages describe each subroutine in detail.

dnet_addr (3dn)

NAME

dnet_addr — convert ASCII node address to binary

SYNTAX

```
#include <netdnet/dn.h>

struct dn_naddr *
dnet_addr (cp)

char *cp;
```

where

Input Argument:

cp is a character pointer to the ASCII node address string. A DECnet node address is specified as *a.n*, where *a* is the area number and *n* is the node number.

A DECnet node address includes an area number (which identifies a node's area in a multiple area network) and a node number (which uniquely identifies a DECnet node). In a multiple area network, *a* is the area number for that node. For a node in a single area network, the *a* argument defaults to 1.

Return Argument:

dn_naddr specifies the node address structure. The following fields are filled in by this subroutine:

a_len specifies the length of the returned node address.

a_addr specifies the node address.

DESCRIPTION

The **dnet_addr** subroutine converts an ASCII node address string to binary and returns a pointer to a **dn_naddr** structure, which contains the node address and the length of the returned node address. This information is required for the **sockaddr_dn** data structure.

RETURN VALUE

If the call succeeds, a pointer to a **dn_naddr** structure is returned. If an error occurs, a value of 0 is returned.

RESTRICTION

If you plan to call this function again before you finish using the data, you must copy the data into a local structure.

dnet_conn (3dn)

NAME

dnet_conn — connect to target network object

SYNTAX

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdnet/dn.h>

int
dnet_conn (node,object,type,opt_out,opt_outl,opt_in,opt_inl)

char *node;
char *object;
u_char *opt_out,*opt_in,
int opt_outl,*opt_inl;
```

where

s is a returned socket descriptor over which a connection has been established.

node specifies the address of the string that contains the remote node name and any optional access data. The node string can have one of the following formats:

"*nodename*[*/username/password/account*]"

or

"*a.n*[*/username/password/account*]"

where *a* is the area number and *n* is the node number.

Node names are matched without regard to case, and access-control information is passed as supplied. (Case is preserved.)

NOTE

Programs that use **dnet_conn** prompt you for a password if you omit the *password* field in an access-control string. The password that you type after the prompt does not echo, which provides account security.

object	specifies the address of the string that contains the target network object. You can specify the object by name or number. If the object number is zero, you must specify the object by name. If the object number is something other than zero, you can specify the object by name or number, but for remote non-ULTRIX nodes, it is recommended that you specify them by number. (To specify the object by name on a remote non-ULTRIX node, see the documentation for that operating system.) Specify object name and number strings as follows: By object name: "*objectname*" (For example, "test".) By object number: "*#objectnumber*" (For example, "#17") Case conversion is not performed on object names before they are sent to a destination program.
type	is the socket type. The DECnet domain currently supports the following socket types: • SOCK_STREAM (stream socket). • SOCK_SEQPACKET (sequenced-packet socket). A value of 0 defaults to SOCK_SEQPACKET.
opt_out	specifies the address of the outgoing optional user data buffer. The message can be up to 16 bytes long. If this argument is not required, supply a NULL pointer.
opt_outl	specifies the size of the optional outgoing message. The message can be up to 16 bytes long. If this argument is not required, supply a NULL value.
opt_in	specifies the address of the buffer that will store the optional incoming message. The message can be up to 16 bytes long. If this argument is not required, supply a NULL pointer.
opt_inl	specifies the size of the buffer that will store the optional incoming message. On return, this argument contains the actual size of the optional incoming message. If this argument is not required, supply a NULL pointer.

DESCRIPTION

The **dnet_conn** subroutine establishes a connection to a specified network object on a remote node. Default access-control information is used to validate access privileges. You can override the default access control by supplying optional access-control information. You can also supply optional connection data.

In addition or instead of supplying access-control information, you can request proxy access on the remote node. The *DECnet–ULTRIX Network Management* manual contains a full description of proxy access.

The **dnet_conn** subroutine requests proxy by default. If you do not want outgoing requests to ask for proxy access at the remote systems, your program should clear the global variable *proxy_requested*.

The **dnet_conn** subroutine creates a socket in the DECnet domain and binds a name to the socket. The bound socket name is the respective user's log-in name converted to uppercase. This bound name is used as the source name for the outgoing connection only when a program is running as superuser; otherwise, the user ID in ASCII is used as the source name.

When you write a program using **dnet_conn**, you must subsequently call **nerror** in order to return any DECnet system errors that occur. For example, if **dnet_conn** returns an error, use the **nerror** subroutine to display the DECnet system error.

RETURN VALUE

If the subroutine completes successfully, the socket descriptor is returned. If an error occurs, a value of -1 is returned. Additional error detail will appear in the external variable **errno**.

RESTRICTION

Currently, **dnet_conn** creates a socket in the DECnet domain and binds a name to the socket. The bound socket name is the respective user's log-in name converted to uppercase. This bound name is used as the source name for the outgoing connection only when a program is running as superuser; otherwise the user ID in ASCII is used as the source name.

DIAGNOSTICS

Use **nerror** to map the following standard ULTRIX errors to equivalent DECnet error messages. The DECnet error text is written to a standard error message.

ULTRIX Error	Equivalent DECnet Error Message
[EACCES]	Connect failed, access control rejected
[EADDRINUSE]	Connect failed, insufficient network resources
[EADDRNOTAVAIL]	Connect failed, unrecognized node name
[ECONNREFUSED]	Connect failed, connection rejected by object
[EHOSTDOWN]	Connect failed, local node shutting down
[EHOSTUNREACH]	Connect failed, node unreachable
[EINVAL]	Connect failed, invalid object name format
[EISCONN]	Connect failed, insufficient network resources
[ENAMETOOLONG]	Connect failed, invalid node name format
[ENETDOWN]	Connect failed, remote node shutting down
[ENOBUFS]	Connect failed, insufficient network resources
[ENOSPC]	Connect failed, insufficient network resources
[ESRCH]	Connect failed, unrecognized object
[ETIMEDOUT]	Connect failed, no response from object
[ETOOMANYREFS]	Connect failed, object too busy

SEE ALSO

errors(2)

dnet_eof (3dn)

NAME

dnet_eof — test for end-of-file on a DECnet socket

SYNTAX

```
int
dnet_eof (s)
int s;
```

where

s specifies a DECnet socket.

DESCRIPTION

The **dnet_eof** subroutine tests a DECnet socket to determine if an end-of-file (EOF) condition exists. An EOF on a DECnet socket indicates that it is impossible to read any more data because no more data exists for a **read** operation and the socket is no longer connected.

This subroutine is useful for determining if an EOF condition exists on a DECnet sequenced-packet socket when a **read** operation has returned zero bytes. ULTRIX uses a returned value of zero on a **read** operation to indicate EOF. Since it is always possible to read a zero-length packet on a DECnet sequenced-packet socket, you cannot determine whether you have just read a zero-length packet or reached EOF without using **dnet_eof**.

RETURN VALUE

If **dnet_eof** determines a connection to be in an active state, a value of 0 is returned. If **dnet_eof** determines a connection to be in an inactive state, a nonzero value is returned.

RESTRICTION

Even though zero-length packets may be available for a **read** operation, **dnet_eof** will indicate an EOF condition if the DECnet socket is no longer connected.

SEE ALSO

read(2dn)

dnet_getalias (3dn)

NAME

dnet_getalias — get extended node information

SYNTAX

```
char *
dnet_getalias (alias)
char *alias;
```

where

alias is a character pointer to an alias.

DESCRIPTION

The **dnet_getalias** subroutine searches for a **.nodes** file in your home directory and returns any alias definitions found in that file. The **dnet_getalias** subroutine returns a node name and any default access-control information associated with the node name.

RETURN VALUE

If a node has default access-control information associated with it, the node name, followed by the access data, is returned in the following format:

nodename/username/password/account

If you have a **.nodes** file in your home directory that defines aliases, any alias definition for the specified alias name is returned.

If a matching alias entry is not found, a NULL pointer is returned.

RESTRICTION

If you plan to call this function again before you finish using the data, you must copy the data into a local structure.

dnet_htoa (3dn)

NAME

dnet_htoa — return ASCII node name or node address

SYNTAX

```
#include <netdnet/dn.h>

char *
dnet_htoa (add)

struct dn_naddr *add
```

where

Input Argument:

add specifies a pointer to a structure of the type **dn_naddr**, which contains the node address.

Return Argument:

dn_naddr specifies the node address structure. The following fields are filled in by this subroutine:

a_len specifies the length of the returned node address.

a_addr specifies the node address.

DESCRIPTION

The **dnet_htoa** subroutine searches the node database for a node by address. If the node is found, the ASCII node name string is returned. If the node is not found, the 16-bit binary node address is converted to the ASCII string representation (*area.number*).

RETURN VALUE

If the node name is found, a pointer to the ASCII node name string is returned. Otherwise, the ASCII node address string is returned.

RESTRICTION

If you plan to call this function again before you finish using the data, you must copy the data into a local structure.

dnet_ntoa (3dn)

NAME

dnet_ntoa — convert binary node address to ASCII

SYNTAX

```
#include <netdnet/dn.h>

char *
dnet_ntoa (add)

struct dn_naddr *add;
```

where

Input Argument:

add specifies a pointer to a structure of the type **dn_naddr**, which contains the node address.

Return Argument:

dn_naddr specifies the node address structure. The following fields are filled in by this subroutine:

a_len specifies the length of the returned node address.

a_addr specifies the node address.

DESCRIPTION

The **dnet_ntoa** subroutine converts a 16-bit binary node address to its ASCII string representation (*area.number*).

RETURN VALUE

A pointer to the ASCII string representation of the DECnet node address is returned.

RESTRICTION

If you plan to call this function again before you finish using the data, you must copy the data into a local structure.

dnet_otoa (3dn)

NAME

dnet_otoa — convert DECnet object name or number to ASCII

SYNTAX

```
#include <netdnet/dn.h>

char *
dnet_otoa (dn)

struct  sockaddr_dn *dn;
```

where

dn is the address of a structure **sockaddr_dn**.

DESCRIPTION

Given a **sockaddr_dn** data structure, **dnet_otoa** converts a DECnet object name or number to its ASCII string representation.

RETURN VALUE

A character pointer to the ASCII string representation of the DECnet object name or number is returned.

RESTRICTION

If you plan to call this function again before you finish using the data, you must copy the data into a local structure.

getnodeadd (3dn)

NAME

getnodeadd — return local node address

SYNTAX

```
#include <netdnet/dn.h>

struct dn_naddr *
getnodeadd();
```

where

dn_naddr specifies the node address structure. The following fields are filled in by this subroutine:

a_len	specifies the length of the returned node address.
a_addr	specifies the local node address.

DESCRIPTION

The **getnodeadd** subroutine returns a pointer to a structure of the type **dn_naddr**, which contains the DECnet node address of your local DECnet–ULTRIX node.

RETURN VALUE

If the subroutine is successful, a pointer to a **dn_naddr** structure is returned. If an error occurs, a value of 0 is returned.

RESTRICTION

If you plan to call this function again before you finish using the data, you must copy the data into a local structure.

getnodeent (3dn)

NAME

getnodeent — get node information from network node database

SYNTAX

```
#include <netdnet/dnetdb.h>

struct  nodeent  *
getnodeent()

struct nodeent *
getnodebyname (name)
char *name

struct nodeent *
getnodebyaddr (addr,len,type)
char *addr;
int len,type;

int setnodeent ()

endnodeent()
```

where

name specifies the address of the buffer containing the node name string.

addr specifies the address of the buffer containing the node address string in the form returned by the **dnet_addr** subroutine.

len is the length of the node's address string.

type specifies the address type. This must be specified as AF_DECnet.

getnodeent returns a pointer to a structure of type **nodeent** with the following members:

```
struct nodeent {

  char          *n_name;          /* name of node */
  int           n_addrtype;       /* node address type */
  int           n_length;         /* length of address */
  char          *n_addr;          /* address */
};
```

nodeent specifies the node-address database structure. The following data fields are filled in by this subroutine:

n_name specifies the name of the node.

n_addrtype specifies the returned address type as AF_DECnet.

n_length specifies the length of the address.

n_addr specifies the network address for the node.

DESCRIPTION

Given a node name or node address, the **getnodebyname** and **getnodebyaddr** subroutines, respectively, access the network node database and return node information. Both return a pointer to a **nodeent** structure. This structure contains an entry from the network node database. The returned **nodeent** structure is stored in static memory allocated in the **getnodeent** subroutine. Therefore, to save it, you must copy it to user memory.

The **getnodebyname** and **getnodebyaddr** subroutines search sequentially from the beginning of the database until a matching host name or host address is found, or until the end of the database is found. Node addresses are always arranged in ascending numeric order.

The **setnodeent**, **getnodeent**, and **endnodeent** functions are similar to the **sethostent**, **gethostent**, and **endhostent** functions. They read through the node database and perform functions in the following order:

1. **setnodeent** sets the pointer to the beginning of the database.
2. **getnodeent** reads the next entry in the database.
3. **endnodeent** closes the database.

RETURN VALUE

setnodeent returns a value of 0 if the subroutine completes successfully; if it fails, a value of -1 is returned.

If **getnodeent** completes successfully, the address for the **nodeent** structure is returned. If an error or an EOF occurs, a value of 0 is returned.

DIAGNOSTICS

NULL pointer (0) is returned on EOF or error.

The following error messages can be returned by the database routines **getnodebyname**, **getnodebyaddr**, **getnodeent**, and **setnodeent**:

[EADDRNOTAVAIL]	No such node name in database.
[EFAULT]	Incompatible database version number.
[ENAMETOOLONG]	The node name is too long.
[ENOBUFS]	Insufficient resources to complete request.
[EPROTONSUPPORT]	Type not supported or length not a valid length.

getnodename (3dn)

NAME

getnodename — return local node name

SYNTAX

```
char *
getnodename()
```

DESCRIPTION

The **getnodename** subroutine returns the ASCII string representation of your local DECnet–ULTRIX node name.

RETURN VALUE

If the subroutine is successful, your local DECnet node name is returned. If an error occurs, a value of 0 is returned.

nerror (3dn)

NAME

nerror — produce DECnet error messages

SYNTAX

```
void
nerror (s)
char *s;
```

where

s is the program name, such as **dlogin**, since the value of the *s* argument is the name of the program that incurred the error (as it is with **perror**).

DESCRIPTION

The **nerror** subroutine produces **dnet_conn** error messages by mapping standard ULTRIX errors to the appropriate DECnet error message. The resulting DECnet error text is written to the standard error file. The error number is taken from the external variable **errno**, which is set when errors occur, but is not cleared when nonerroneous calls are made.

The DECnet error text is output to the standard error file.

RETURN VALUE

This function returns no value.

SEE ALSO

dnet_conn (3dn)

Appendix A

DECnet–ULTRIX Data Structures

This appendix shows the DECnet–ULTRIX data structures. For guidelines on specifying these data structures, see the relevant system calls and the header file **/sys/netdnet/dn.h**.

A.1 Access-Control Information Data Structure

```
struct accessdata_dn {
     unsigned short  acc_accl;       /* length of account string */
     unsigned char   acc_acc[40];    /* account string */
     unsigned short  acc_passl;      /* length of password string */
     unsigned char   acc_pass[40];   /* password string */
     unsigned short  acc_userl;      /* length of user string */
     unsigned char   acc_user[40];   /* user string */
};
```

A.2 DECnet Node Address Data Structure

```
# define DN_MAXADDL    2

    struct dn_naddr {
            unsigned short a_len;            /* length of address */
            unsigned char a_addr[DN_MAXADDL]; /* address as bytes */
        };
```

NOTE

The structure member *a_addr* represents the DECnet Phase IV node address. It is a 16-bit unsigned value, where bits 0-9 are the node address and bits 10-15 are the area number.

A.3 Logical Link Information Data Structure

```
struct linkinfo_dn {
       unsigned short  idn_segsize;    /* segment size for link */
       unsigned char   idn_linkstate;  /* logical link state */
};
```

A.4 Optional User Data Structure

```
struct optdata_dn {
        unsigned short  opt_status;     /* extended status return */
        unsigned short  opt_optl;       /* length of user data */
        unsigned char   opt_data[16];   /* user data */
};
```

A.5 Socket Address Data Structure

```
struct sockaddr_dn {
        unsigned short  sdn_family;     /* AF_DECnet */
        unsigned char   sdn_flags;      /* flags */
        unsigned char   sdn_objnum;     /* object number */
        unsigned short  sdn_objnamel;   /* size of object name */
                char    sdn_objname[16];        /* object name */
        struct dn_naddr  sdn_add;       /* node address */
};
```

Appendix B

DECnet–ULTRIX Programming Examples

This appendix presents the following types of programming examples:

- A sample client program using **dnet_conn**
- A sample server program using the **dnet_spawner**
- A sample client program using system calls
- A sample server program using system calls
- A sample gateway program

These programming examples are also available on line in **/usr/examples/decnet**.

B.1 Sample Client Program Using dnet_conn

```
#ifndef lint
static  char  *sccsid = "@(#)dnet_echo1.c       1.5  7/27/90";
#endif lint

/*
 *
 * d e c n e t  e x a m p l e : d n e t  e c h o 1
 *
 * Description: This client program connects to the partner server
 *              program "dnet_echo1d" on the node specified on the
 *              command line.  Once connected, lines are read from
 *              standard input and shipped over the network to
 *              dnet_echo1d, which then echoes the lines back to
 *              this program, which then prints them on standard
 *              output.
 *
 * Input:       Name of the node on which dnet_echo1d will run.
 *
 * Output:      none
 *
 * To compile:  cc dnet_echo1.c -ldnet -o dnet_echo1
 *
 * Examples:    dnet_echo1 boston
 *              dnet_echo1 boston/jones/topsecret
 *
 * Comments:    This example illustrates the use of dnet_conn to
 *              establish a network connection.   Compare the
 *              simplicity of using dnet_conn with doing the connect
 *              with system calls, as shown in example dnet_echo2.c.
 *              Whether access control is required depends on how the
 *              companion program (dnet_echo1d) was set up.
 */

/*
 * Digital Equipment Corporation supplies this software example on
 * an "as-is" basis for general customer use.  Note that Digital
 * does not offer any support for it, nor is it covered under any
 * of Digital's support contracts.
 */

#include <stdio.h>

#define BUFSIZE 1024        /* size of buffer for read/write */

main(argc, argv)
int argc;
char *argv[];
{
    int sock;              /* socket for connection */
    int length;            /* length of data        */
    char buff[BUFSIZE];    /* buffer for data       */

    /*
     * Make sure the node name was given on the command line
     */
    if( argc < 2 ) {
        fprintf(stderr, "Usage: %s nodename\n", argv[0]);
        exit();
    }
```

```
        /*
         * connect to our partner "dnet_echo1d" on the specified node
         */
        sock = dnet_conn(argv[1], "dnet_echo1d", 0, 0, 0, 0, 0);
        if( sock < 0 )
        {
            /* print DECnet specific connect error */
            nerror (argv[0]);
                    exit();
        }

        puts("Connected!");

        /*
         * read lines from standard input, send them over the network,
         * then read and print the echoed line from our partner
         */
        while( gets(buff) != NULL )
        {
            length = strlen(buff);

            /* Only send nonempty lines */
            if( length > 0 ) {
                if( write(sock, buff, length) < 0 ) {
                    perror("couldn't send line over network");
                    break;
                }
                if( (length = read(sock, buff, BUFSIZE)) < 0 ) {
                    perror("couldn't read line over network");
                    break;
                }

                buff[length] = '\0';
                puts(buff);
            }
        }
        /*
         * finished - close network connection and exit
         */
        puts("Exiting...");
        close(sock);
}
```

B.2 Sample Server Program Using the dnet_spawner

```
#ifndef lint
static  char  *sccsid = "@(#)dnet_echo1d.c      1.5  7/27/90";
#endif lint

/*
 *
 * d e c n e t   e x a m p l e : d n e t   e c h o 1 d
 *
 * Description: This server program reads messages from  the network
 *              connection and then writes(echoes) them back to the
 *              network connection, until the network connection is
 *              broken.
 *
 * Input:       none
 *
 * Output:      none
 *
 * To compile:  cc dnet_echo1d.c -ldnet -o dnet_echo1d
 *
 * Comments:    This example illustrates the use of the dnet_spawner
 *              to listen for incoming connect requests on behalf of
 *              other server programs.  Note that standard in and
 *              standard out are directed to the network connection
 *              (socket) by the dnet_spawner before executing this
 *              program. Compare the simplicity of using the spawner
 *              with the complexity of using system calls as is done
 *              in example dnet_echo2d.c.
 *
 *              To work with the spawner the decnet object database
 *              must be properly configured. There are two
 *              alternatives:
 *                1)  A person with superuser privileges must define
 *                    this object using ncp, for example:
 *                    ncp set object dnet_echo1d \
 *                        file /usr/examples/dnet_echo1d
 *                2)  Object "DEFAULT" must be defined in the object
 *                    database (DECnet comes with DEFAULT defined),
 *                    and this program must be moved to a directory
 *                    searched by the spawner (for example, /usr/bin).
 *              In either case, if no default user is defined for the
 *              object, access control information must be specified
 *              by the client (dnet_echo1) when attempting to
 *              connect.  If a default user is defined, and such an
 *              account actually exists, then access control is not
 *              required (although it can still be specified if
 *              desired).
 */

/*
 * Digital Equipment Corporation supplies this software example on
 * an "as-is" basis for general customer use.  Note that Digital
 * does not offer any support for it, nor is it covered under any
 * of Digital's support contracts.
 */

#include <stdio.h>

#define BUFSIZE 1024   /* size of buffer for read/write */

#define TRUE    1
#define FALSE   0

main()
{
    char buff[BUFSIZE];
    int length;
```

```
        while( TRUE )
    {
        /*
         * read messages from standard input,
         * write them to standard output
         */
        length = read(0, buff, sizeof(buff));
        if( length <= 0 )
             /* if at "end-of-file" (connection broken) */
            if( dnet_eof(0) )
                break;

        write(1, buff, length);
    }
}
```

B.3 Sample Client Program Using System Calls

```
#ifndef lint
static  char  *sccsid = "@(#)dnet_echo2.c       1.5  7/27/90";
#endif lint

/*
 *
 * d e c n e t  e x a m p l e : d n e t  e c h o 2
 *
 * Description: This client program connects to the partner server
 *              program "dnet_echo2d" on the node specified on the
 *              command line.  Once connected, lines are read from
 *              standard input and shipped over the network to
 *              dnet_echo2d, which then echoes the lines back to
 *              this program, which then prints them on standard
 *              output.
 *
 * Input:       Name of the node on which dnet_echo2d will run.
 *
 * Output:      none
 *
 * To compile:  cc dnet_echo2.c -ldnet -o dnet_echo2
 *
 * Example:     dnet_echo2 boston
 *
 * Comments:    This example illustrates the use of system calls
 *              to establish a network connection.  Compare this
 *              method with that of using dnet_conn as is done in
 *              dnet_echo1.c.  Also, the connect request is by
 *              object number, rather than by object name as was
 *              done in dnet_echo1.c.
 */

/*
 * Digital Equipment Corporation supplies this software example on
 * an "as-is" basis for general customer use.  Note that Digital
 * does not offer any support for it, nor is it covered under any
 * of Digital's support contracts.
 */

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdnet/dn.h>
#include <netdnet/dnetdb.h>

#define BUFSIZE 1024        /* size of buffer for read/write */

main(argc, argv)
int argc;
char *argv[];
{
    int sock, length;
    char buff[BUFSIZE];
    struct sockaddr_dn address;
    struct dn_naddr *node_addr;
    struct nodeent *nodep;

    if( argc < 2 ) {
        fprintf(stderr, "Usage: %s nodename\n", argv[0]);
        exit();
    }

    /* Create a socket in DECnet domain */
    if( (sock = socket(AF_DECnet, SOCK_SEQPACKET, 0)) < 0 ) {
        perror(argv[0]);
        exit();
    }
```

```
        /* Specify target object number for connection */
        bzero(&address, sizeof(address));
        address.sdn_family = AF_DECnet;
        address.sdn_objnum = 128;

        if( (node_addr = dnet_addr(argv[1])) == NULL ) {
            if( (nodep = getnodebyname(argv[1])) == NULL ) {
                fprintf(stderr, "%s: Node unknown\n", argv[1]);
                exit();
            }
            else {
                bcopy(nodep->n_addr,address.sdn_nodeaddr,nodep->n_length);
                address.sdn_nodeaddrl = nodep->n_length;
            }
        }
        else
            address.sdn_add = *node_addr;

        /* Connect to partner on specified node */
        if ( connect(sock, &address, sizeof(address)) < 0 ) {
            perror(argv[0]);
            exit();
        }
        puts("Connected...");

        /* Read lines from standard input, send them over */
        /* the network connection, and print the response */
          while( gets(buff) != NULL )
        {
            length = strlen(buff);

            /* Only send non-empty lines */
            if( length > 0 ) {
                if( write(sock, buff, length) < 0 ) {
                    perror("couldn't send line over network");
                    break;
                }

                if( (length = read(sock, buff, BUFSIZE)) < 0 ) {
                    perror("couldn't read line over network");
                    break;
                }

                buff[length] = '\0';
                puts(buff);
            }
        }

        printf("Exiting...\n");      /* Close link and exit */
        close(sock);
    }
```

B.4 Sample Server Program Using System Calls

```
#ifndef lint
static  char  *sccsid = "@(#)dnet_echo2d.c      1.5  7/27/90";
#endif lint

/*
 *
 * d e c n e t   e x a m p l e : d n e t   e c h o 2 d
 *
 * Description: This server program is designed to run as a daemon.
 *              When a network connection is created, it forks and
 *              executes a child, which then reads messages from the
 *              network connection and then writes(echoes) them back
 *              to the network connection, until the network
 *              connection is broken.
 *
 * Input:       none
 *
 * Output:      none
 *
 * To compile:  cc dnet_echo2d.c -ldnet -o dnet_echo2d
 *
 * Example:     dnet_echo2d &
 *
 * Comments:    This example illustrates the programming of a server
 *              that does not use the dnet_spawner.  This program
 *              must be started manually before attempting to connect
 *              to it from the client program (dnet_echo2).  Note
 *              that no access control verification is done in this
 *              example, but a "real" server program would
 *              need to do some form of access control verification
 *              (in example dnet_echo1d, access control verification
 *              is done automatically by the dnet_spawner).
 */

/*
 * Digital Equipment Corporation supplies this software example on
 * an "as-is" basis for general customer use.  Note that Digital
 * does not offer any support for it, nor is it covered under any
 * of Digital's support contracts.
 */

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdnet/dn.h>
#include <sys/wait.h>
#include <signal.h>
#include <errno.h>

#define BUFSIZE 1024        /* size of buffer for read/write */

main(argc, argv)
int argc;
char *argv[];
{
    int s, ns, acclen, rdlen;
    char buf[BUFSIZ];
    struct sockaddr_dn sockaddr, accsockaddr;

    /* Create socket in DECnet address family */
    /* of type sequenced packet.              */
    if ( (s = socket(AF_DECnet, SOCK_SEQPACKET, 0)) < 0 ) {
        perror(argv[0]);
        exit();
    }
```

```
    /* The socket address indicates the DECnet address */
    /* and object number of 128.                       */
    bzero(&sockaddr, sizeof(struct sockaddr_dn));
    sockaddr.sdn_family = AF_DECnet;
    sockaddr.sdn_objnum = 128;

    /* Bind the socket to a DECnet socket address and */
    /* listen for a connection.                       */
    if( bind(s, &sockaddr, sizeof(sockaddr)) < 0 ) {
        perror(argv[0]);
        exit();
    }

    if( listen(s, SOMAXCONN) < 0 ) {
        perror(argv[0]);
        exit();
    }

    /* Accept an incoming connection */
    for( ; ; ) {
        do {
            acclen = sizeof(accsockaddr);
            ns = accept(s, &accsockaddr, &acclen);
        } while( ns == -1  &&  errno == EINTR );

    /* Fork child to handle the new connection */
    if( fork() == 0 )
        break;
    close(ns);
    }

    /* Redirect standard input and output to new socket */
    dup2(ns, 0); dup2(ns, 1);
    close(ns); close(s);

    for( ; ; ) {
    if( (rdlen = read(0, buf, sizeof(buf))) <= 0 )
        if( dnet_eof(0) )
            break;

    write(1, buf, rdlen);
    }
}
```

B.5 Sample Application Gateway Program

```
#ifndef lint
static  char  *sccsid = "@(#)gatewayd.c         1.4  5/27/90";
#endif lint

/*
 * DESCRIPTION
 *
 *   This program illustrates how an ULTRIX system can be used as a
 *   gateway to swap transports for an application protocol.  A brief
 *   description of how the program is used is given below.  For more
 *   details on usage, see file /usr/examples/decnet/gatethru/README
 *
 * USAGE
 *
 *   gatewayd
 *   gatewayd -inet desthost destservice
 *   gatewayd -dnet destnode destobject
 *
 *   In the first case (with no arguments specified), three
 *   lines should be sent initially:
 *
 *           protocol      ("inet" or "dnet" without the quotes)
 *           destsystem    (host or node name)
 *           destentity    (service or object name)
 *
 *           These must be delimited by one of the following:
 *
 *              <LF>       (linefeed)
 *              <CR><LF>   (carriage-return linefeed)
 *
 *           A response will be returned as:
 *
 *              Connected to destsystem (destentity) via protocol
 *
 *           for success, and
 *
 *              Not-Connected [further explanation]
 *
 *           for failure.  These responses will be delimited by the
 *           same delimiter that had delimited the protocol.
 *
 *   In the other cases, there will be no exchange.  If the
 *   connection couldn't complete to the destsystem/destentity,
 *   the connection to the client is simply disconnected.
 *
 *   In all cases, "service" must be defined in /etc/services on the
 *   gateway system, and "host" must be in /etc/hosts.
 */

/*
 * Digital Equipment Corporation supplies this software example on
 * an "as-is" basis for general customer use.  Note that Digital
 * does not offer any support for it, nor is it covered under any
 * of Digital's support contracts.
 */

#include <stdio.h>
#include <strings.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/ioctl.h>
#include <netdnet/dn.h>
#include <netinet/in.h>
#include <netdb.h>

#include <signal.h>
```

```
#include <errno.h>
#include <sysexits.h>

#define      STREQL(a, b)        (strcmp(a, b) == 0)
#define      NIL                 (0)

char      DestProto[40];     /* Protocol family to connect by       */
char      DestHost[256];     /* Remote system to connect to         */
char      DestObj[256];      /* Remote object/service to connect to */

char      LineDelim[10] = "\n"; /* Default line delim in smart mode */

int      SmartMode;         /* Get destination info from client mode */

/* Signal handler for SIGPIPE (write on a disconnected socket) */
abort()
{
    /* We still had data to transfer and the side who should have
       received it has gone away.  We will consider it an I/O error */
    exit(EX_IOERR);
}

main(argc, argv)
int    argc;            /* # of command line arguments */
char   *argv[];         /* the command line arguments  */
{
    int      client,       /* Socket connected to client  */
             server;       /* Socket to use for server    */

    /* Check usage */
    if( !(argc == 4  ||  argc == 1) )
      exit(EX_USAGE);

    /* If no arguments, operate in Smart Mode */
    SmartMode = argc == 1;

    if( !SmartMode ) {
      /* Fetch connect info from command line */
      strcpy(DestProto, &argv[1][1]);
      strcpy(DestHost,   argv[2]);
      strcpy(DestObj,    argv[3]);
    }
    else {
      char *p;        /* Temp */

      /* Get connect info from client */
      fgets(DestProto, sizeof(DestProto), stdin);
      fgets(DestHost,  sizeof(DestProto), stdin);
      fgets(DestObj,   sizeof(DestProto), stdin);

      p = strpbrk(DestProto, "\r\n");
      strcpy(LineDelim, p);
      *p = '\0';

      strpbrk(DestHost, "\r\n")[0] = '\0';
      strpbrk(DestObj,  "\r\n")[0] = '\0';
    }

    /* Time to attempt the connection */

    if( STREQL(DestProto, "dnet") ) {
      server = dnet_conn(DestHost, DestObj, SOCK_STREAM,
                         (u_char *)0, 0, (u_char *)0, (int *)0);

      if( server < 0 ) {
          /* Return failure indication back to client in Smart Mode */
          if( SmartMode ) {
              printf("Not-Connected%s", LineDelim);
              fflush(stdout);
          }
```

```
            /* Some spawners (DECnet) log children's exit codes */
            exit(EX_CANTCREAT);
        }
    }
    else if( STREQL(DestProto, "inet") ) {
        server = inet_conn(DestHost, DestObj);

        if( server < 0 ) {
            /* Return failure indication back to client in Smart Mode */
            if( SmartMode ) {
                printf("Not-Connected%s", LineDelim);
                fflush(stdout);
            }

            /* Some spawners (DECnet) log children's exit codes */
            exit(EX_CANTCREAT);
        }
    }
    else {
        /* Error; Request to connect via an unknown protocol */

        /* Return failure indication back to client in Smart Mode */
        if( SmartMode ) {
            printf("Not-Connected Unknown protocol %s%s",
                    DestProto, LineDelim);
            fflush(stdout);
        }

        /* Some spawners (DECnet) log children's exit codes */
        exit(EX_PROTOCOL);
    }

    /* Return success indication back to client in Smart Mode */
    if( SmartMode ) {
        printf("Connected to %s (%s) via %s%s",
            DestHost, DestObj, DestProto, LineDelim);
        fflush(stdout);
    }

    /* Just to make the code more readable */
    client = 0;

    /* We will abort gracefully when the client or remote system
       goes away */
    signal(SIGPIPE, abort);

    /* Now just go and move raw data between client and
       remote system */
    dowork(client, server);
    /* ... NEVER RETURNS ... */
}

dowork(client, server)
int    client, server;
{

/* select(2) masks for client and remote */
int      ClientMask, ServerMask;

/* Combined ClientMask and ServerMask */
int      ReadMask;

    /* Initialize select(2) masks */
    ClientMask = 1<<client;
    ServerMask = 1<<server;

    ReadMask = ClientMask | ServerMask;

    /* Now move raw data for the rest of our life between
       client and remote */
    for( ; ; ) {
        /* Local Variables */
        int  SelectReadMask;/* select(2) mask modifiable by select(2) */
        int  nready;        /* status return from select(2)           */
```

```
        do {
            /* Intialize select(2) mask every time
               as select(2) always modifies it */
            SelectReadMask = ReadMask;

            /* Wait for data to be present to be moved */
            nready = select(32,&SelectReadMask,(int *)0,(int *)0,NIL);
        } while( nready < 0  &&  errno == EINTR );

        /* select(2) failed, should not happen.  Exit abnormally */
        if( nready < 0 )
            exit(EX_SOFTWARE);

        /* Favor the client (unspecified reason)
           if s/he has data */
        if( SelectReadMask & ClientMask )
            xfer(client, server);

        /* Then check on the other operation*/
        if( SelectReadMask & ServerMask )
            xfer(server, client);
    }

    /* NEVER REACHED */
}

#define      BUFSIZE        256 /* Max bytes to move at a time */

xfer(from, to)
int      from, to;        /* Move data from "from" to "to" */
{
static char buf[BUFSIZE];      /* Buffer data to be moved      */
int      nready;               /* # bytes readable             */
int      got;                  /* # bytes actually being moved */

    /* Query the system how many bytes are ready to be read */
    ioctl(from, FIONREAD, &nready);

    /* Only try to get the smaller of nready and BUFSIZE */
    got = read(from, buf, nready < BUFSIZE ? nready : BUFSIZE);

    /* Zero bytes returned indicates end of stream, exit gracefully */
    if( got == 0 )
      exit(EX_OK);

    /* Now send it across to the other side */
    write(to, buf, got);
}

int
inet_conn(host, port)
    char *host;
    char *port;
{
/* Local Vars */
int                sock;      /* Socket to use for the connection */
struct hostent     *hostent;  /* Destination host entry           */
struct servent     *servent;  /* Destination service entry        */
struct sockaddr_in addr;      /* Formatted destination for connect */

    /* Fetch the requested host and service entries */
    hostent = gethostbyname(host);
    servent = getservbyname(port, "tcp");

    /* No host entry, no service entry, or host is not
       Internet, error! */
    if( servant == NIL ||
      hostent == NIL ||
      hostent->h_addrtype != AF_INET )
      return -1;

    /* Get a socket from the system to use for the connection */
    if( (sock = socket(AF_INET, SOCK_STREAM, 0)) < 0 )
      return -1;
```

```
    /* Make sure we start with a clean address structure ... */
    bzero(&addr, sizeof(addr));

    /* ... then fill in the required fields */
    addr.sin_family = AF_INET;
    addr.sin_port   = servant->s_port;
    bcopy(hostent->h_addr, &addr.sin_addr, hostent->h_length);

    /* Now try connection to the destination */
    if( connect(sock, &addr, sizeof(addr)) < 0 ) {
      /* No go, release the socket, and then return error! */
      close(sock);
      return -1;
    }

    /* Success.  Return the connected socket descriptor */
    return sock;
}
```

Glossary

Accept-Deferred Mode

A mode for accepting incoming connections. **Deferred mode** lets the server program store, examine, and process any access control information or optional data that is supplied as part of a connection request.

Accept-Immediate Mode

A mode for accepting incoming connections. **Immediate mode** makes it possible for the server program to send and receive data as soon as the accept call operation completes.

Access Control Information

Identification information used to screen inbound connect requests and verify them against a system account. In the DECnet domain, access control information consists of a specified user name, password, and account string.

Blocking Input/Output (I/O)

An I/O mode that causes a calling process to wait for an input/output operation. Blocking prevents an input/output system call from returning control to a calling procedure until the operation completes. See also **Nonblocking Input/Output**.

Client Application

Any application that initiates a connection and requests services from the server application.

Client-Server Communication

Task-to-task communication between applications through a socket interface.

Communication Domain

A set of protocols that have common communication properties. For example, the Internet domain supports applications that communicate through the Defense Advanced Research Projects Agency (DARPA) standard communication protocols, and the DECnet domain supports applications that communicate through the Digital Network Architecture.

Digital Data Communications Message Protocol (DDCMP)

A set of conventions used for data transmission over physical links.

Interprocessor Communication (IPC)

Communication between two independent processes, such as client and server programs. These processes use system calls to establish connections and communicate with each other through sockets.

Network Object

A task or program (for example, **fal** or **nml**) that provides generic services across a network. In the DECnet–ULTRIX programming environment, a network object is a server application that can be accessed from other Internet or DECnet nodes on a network.

Nonblocking Input/Output (I/O)

A mode that causes a calling process to not wait for an I/O operation. The nonblocking input/output mode returns control to the calling procedure immediately with an error message if there are not enough resources available to complete the operation. See also **Blocking Input/Output**.

Optional data

In the DECnet domain, a string of up to 16 bytes that clients and servers can exchange on either a connect or disconnect sequence. This data is interpreted differently according to the application.

Out-of-Band Message

An unsolicited, high-priority message that one application sends to another outside of the normal data channel. In most cases, it informs the receiving application of an unusual or abnormal event in the sending application.

Proxy Access

A method of screening client application access to a server application without using a password. The supplied name of the user making the access request must correspond with an entry listed in the target node's proxy access file.

Sequenced-Packet Socket

A socket type that preserves record boundaries and supplies a bidirectional, reliable, ordered, first-in, first-out (FIFO), unduplicated flow of data.

Server Application

Any application that either accepts or rejects a request from a client application and provides services to client applications.

Stream Socket

A socket type that provides a byte stream without using message boundaries. It also supplies a bidirectional, reliable, ordered, first-in, first-out (FIFO), unduplicated flow of data.

Socket

An addressable endpoint for communication. Client and server applications each create a socket that acts as a handle for sending and receiving data.

Index

L

M

N

O

P

R

S

W

HOW TO ORDER ADDITIONAL DOCUMENTATION

DIRECT TELEPHONE ORDERS

In Continental USA
call 800–DIGITAL

In Canada
call 800–267–6215

In New Hampshire
Alaska or Hawaii
call 603–884–6660

In Puerto Rico
call 809–754–7575
x2012

ELECTRONIC ORDERS (U.S. ONLY)

Dial 800–DEC–DEMO with any VT100 or VT200
compatible terminal and a 1200 baud modem.
If you need assistance, call 1–800–DIGITAL.

DIRECT MAIL ORDERS (U.S. and Puerto Rico*)

DIGITAL EQUIPMENT CORPORATION
P.O. Box CS2008
Nashua, New Hampshire 03061

DIRECT MAIL ORDERS (Canada)

DIGITAL EQUIPMENT OF CANADA LTD.
940 Belfast Road
Ottawa, Ontario, Canada K1G 4C2
Attn: A&SG Business Manager

INTERNATIONAL

DIGITAL
EQUIPMENT CORPORATION
A&SG Business Manager
c/o Digital's local subsidiary
or approved distributor

Internal orders should be placed through the Software Distribution Center (SDC),
Digital Equipment Corporation, Westminster, Massachusetts 01473

*Any prepaid order from Puerto Rico must be placed
with the Local Digital Subsidiary:
809–754–7575 x2012

HOW TO ORDER ADDITIONAL DOCUMENTATION

DIRECT TELEPHONE ORDERS

In Continental USA
and 800-DIGITAL

In Canada
call 800-267-6215

In New Hampshire
Alaska or Hawaii
call 603-884-6660

In Puerto Rico
call 809-754-7575
x2010

ELECTRONIC ORDERS (U.S. ONLY)

Dial a D-TIE-DECNET node with a VT100 or VT200
compatible terminal and a 1200 baud modem.
If you need assistance, call 1-800-DIGITAL

DIRECT MAIL ORDERS (U.S. and Puerto Rico)

DIGITAL EQUIPMENT CORPORATION
P.O. Box CS2008
Nashua, New Hampshire 03061

DIRECT MAIL ORDERS (Canada)

DIGITAL EQUIPMENT OF CANADA LTD
940 Belfast Road
Ottawa, Ontario, Canada K1G 4C2
Attn: A&SG Business Manager

INTERNATIONAL

DIGITAL
EQUIPMENT CORPORATION
A&SG Business Manager
c/o Digital's local subsidiary
or approved distributor

For single copies of documents, send your name, company name, printed address, and the document title, order number, and quantity to the Publications Group at the above address.

A U.S. customer may charge orders of $50 or more to its VISA or MasterCard.
Mail orders must be prepaid. Shipping and handling charges will be added.
Prices are subject to change.

DECnet–ULTRIX
Programming
AA–EA88D–TE

READER'S COMMENTS

What do you think of this manual? Your comments and suggestions will help us to improve the quality and usefulness of our publications.

Please rate this manual:

	Poor				Excellent
Accuracy	1	2	3	4	5
Readability	1	2	3	4	5
Examples	1	2	3	4	5
Organization	1	2	3	4	5
Completeness	1	2	3	4	5

Did you find errors in this manual? If so, please specify the error(s) and page number(s).

_____

_____

_____

_____

General comments:

_____

_____

_____

_____

Suggestions for improvement:

_____

_____

_____

_____

Name _____ Date _____

Title _____ Department _____

Company _____ Street _____

City _____ State/Country _____ Zip _____

DO NOT CUT – FOLD HERE AND TAPE

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY LABEL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

digital™

Networks and
Communications Publications
550 King Street
Littleton, MA 01460–1289

DO NOT CUT – FOLD HERE

CUT ON THIS LINE

digital

- Each object known to the network and to the local node

1.4.1 Displaying Volatile and Permanent Databases

Before you modify a network component, you must use a different command to display its parameter values, depending on its location. If it is in the permanent database, use the **list** command to display parameter values for each component. If it is in the volatile database, use the **show** command.

See Chapter 4 for examples of how to display information about each component.

1.4.2 Modifying Volatile and Permanent Databases

You use **ncp** commands to modify parameter values defined in either database for any of these network components. You use different commands to configure network components in the volatile database than in the permanent database.

The **define** and **purge** commands act on the permanent database and take effect when the network is restarted, whereas their functional counterparts, the **set** and **clear** commands, act immediately on the volatile (running) database.

To modify values in the permanent database, use the following commands:

define	adds or changes a parameter value.
purge	removes parameter values.

To modify values in the volatile database, use the following commands:

set	changes or resets a parameter value.
clear	removes parameter values.

Changes you make to the volatile database go into effect immediately. When you start up the system, the initial version of the volatile database is a copy of the permanent database.

If you alter the volatile database, you can restore the parameter values from the permanent database for any component by using the **all** parameter with a **set** command for that component. To display the current parameter values in the volatile database, use the **show** command.

Changes you make using the **set** and **clear** commands remain in effect until you make another change, restart DECnet–ULTRIX software on the local system, or reboot the operating system. When you reload, the system configuration matches the parameter values in the permanent database.

1.5 Privileges

On ULTRIX systems, you need superuser privileges to execute **ncp** commands that change a database. Other Digital systems may also require privileges for the same **ncp** commands. To determine the privileges you need to issue **ncp** commands at a DECnet–ULTRIX node for remote execution, see the network management documentation for that remote node.

1.6 Remote Access

You can issue **ncp** commands for execution at either the local node or a remote node. If a remote non-ULTRIX node supports an expanded set of **ncp** commands, you can execute the full set of **ncp** commands on that node. (See the remote system's documentation for a list of supported commands.)

For more information about remote access, see the *DECnet–ULTRIX NCP Command Reference* manual.

1.7 Down-Line Loading a Remote Node

The Network Control Program gives you down-line loading capabilities so that you can boot a remote node from your local node. You can down-line load a remote node from your local node by issuing **ncp load** and **trigger** commands.

You can also specify parameters on the **ncp load node** and **ncp trigger node** command lines to override current parameter values in the load host's database.

NOTE

Before you can use the **ncp load** and **trigger** commands, the ULTRIX **mop_mom** utility must be installed on your system. For further information, see **mop_mom** in the ULTRIX documentation set.

The following sections explain how to use **load** and **trigger** commands to initiate a down-line load from a load host.

1.7.1 Using the ncp load Command

If you use the **ncp load** command, you must issue it at a load host. The **ncp load** command ensures that the load host at which you issue the command is the node that performs the down-line load.

After you issue the **load** command on the load host, the down-line load proceeds as follows:

1. The load host sends a **mop remote console boot** message with a direct load option specified.
2. When the remote node receives this message, it sends a **mop request program** message directly to that load host.
3. The load host and the remote node use additional **mop** messages to transfer the remote node's software image into the remote node's memory.

1.7.2 Using the trigger Command

If you use the **ncp trigger** command, you can issue it from any supported network management host. Because it does not have to wait for a specific load host to respond to a load request, the **trigger** command is usually faster than the **load** command. The **trigger** command does not specify which load host performs the down-line load. Therefore, you may not know beforehand which load host will actually down-line load the load host image. A **syslog** message informs you after

DECnet–ULTRIX

Network Management

May 1990

This manual introduces DECnet databases and components and shows how you use the Network Control Program (ncp) to configure, monitor, and test the components.

Supersession/Update Information:	This is a revised manual.
Operating System and Version:	ULTRIX V4.0
Software Version:	DECnet–ULTRIX V4.0

digital™

Order Number: AA–EE38D–TE

AA–EE38D–TE
May 1990

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by Digital or its affiliated companies.

Restricted Rights: Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227–7013.

The following are trademarks of Digital Equipment Corporation:

DEC	PDP	VAX
DECnet	ULTRIX	VMS
DECUS	UNIBUS	digital™

UNIX is a registered trademark of AT&T in the USA and other countries.

This manual was produced by Networks and Communications Publications.

Contents

Preface

This manual shows you how to manage a DECnet–ULTRIX node in the DECnet environment. DECnet–ULTRIX software works with systems running ULTRIX software and conforms to the Digital Network Architecture (DNA). DNA, the model for all DECnet implementations, allows all Digital operating systems to participate in the same network.

Manual Objectives

This manual describes the network databases and components and shows you how to configure, monitor, and test network components using the Network Control Program (NCP). For detailed descriptions of the **ncp** commands and error messages, refer to the *DECnet-ULTRIX NCP CommandReference* manual. Other topics include loopback testing, event logging, and displaying system counters and database information.

Intended Audience

This manual is for the network manager, the person responsible for configuring, managing, and maintaining the network.

Structure of This Manual

This manual contains six chapters:

Chapter 1	Gives an overview of DECnet–ULTRIX network management. Introduces the configuration databases and the tools for managing them.
Chapter 2	Describes basic network components and tells you how to use **ncp** commands to configure them.
Chapter 3	Explains how to control access by using access-control information and setting up proxy.
Chapter 4	Tells how to monitor the network using display commands, network counters, event logging, and object log files.
Chapter 5	Describes node-level and circuit-level loopback tests, and **dts/dtr** tests.
Chapter 6	Contains quick-reference manual pages for the DECnet–ULTRIX network maintenance commands **ncp** and **dts/dtr**.

Chapter 1

Overview of the DECnet Network

Before you perform network management tasks, you should already be familiar with the DECnet network components, configuration databases, and network management tools.

This chapter briefly overviews DECnet structure, components, and configuration databases. Other topics include tools to configure and control the network, and basic network management tasks.

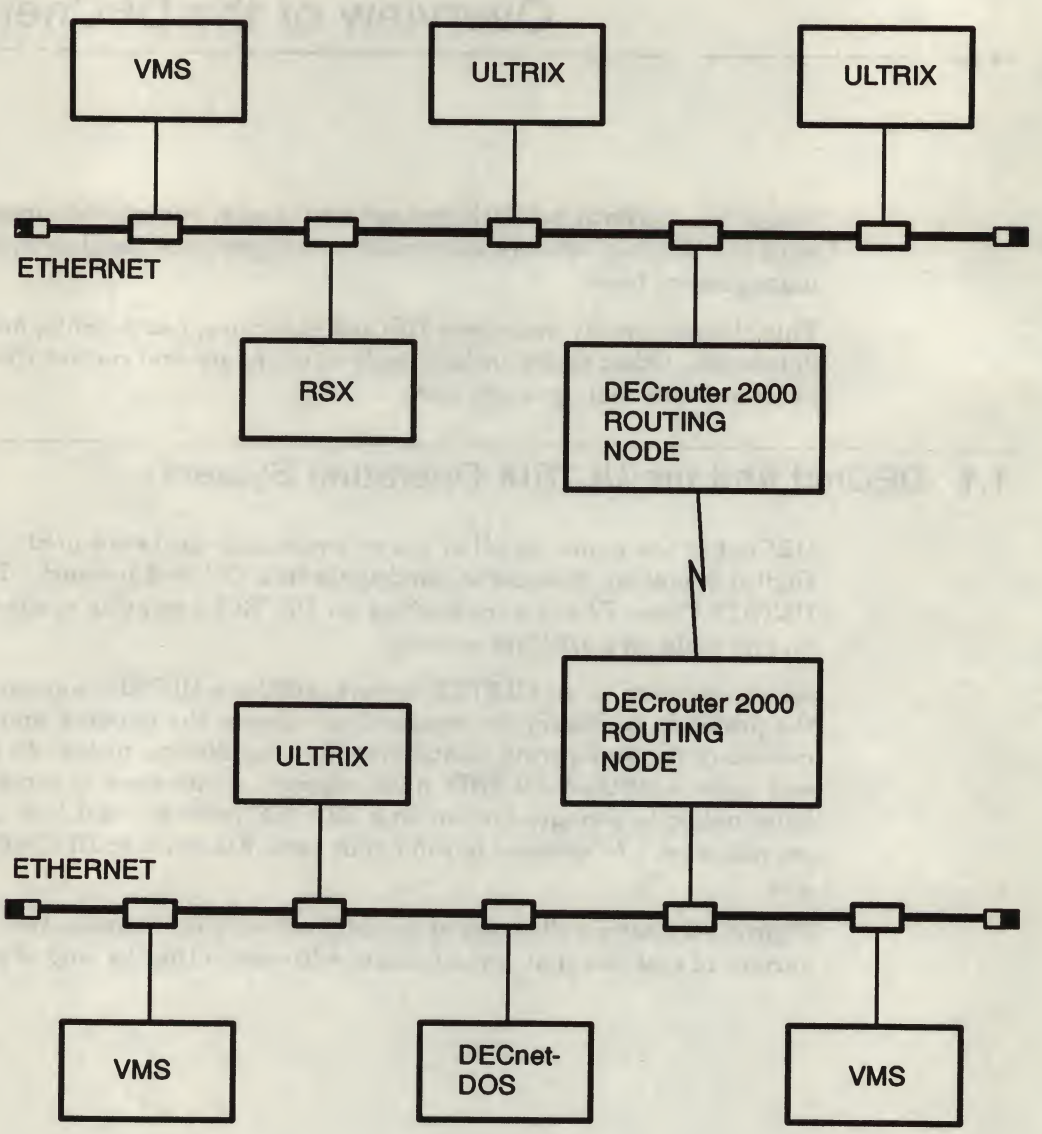
1.1 DECnet and the ULTRIX Operating System

DECnet is the name for all of the software and hardware products that enable Digital operating systems to participate in a DECnet network. The DECnet–ULTRIX Phase IV software enables an ULTRIX operating system to function as an end node on a DECnet network.

As an interface to an ULTRIX system, DECnet–ULTRIX software supports the protocols necessary for communicating over the network and the functions necessary for configuring, controlling, and monitoring nodes. As a DNA Phase IV end node, a DECnet–ULTRIX node supports connections to other systems on an Ethernet or to a single system on a DDCMP point-to-point line. Routing nodes provide access to systems beyond your local Ethernet or DDCMP point-to-point line.

Figure 1-1 shows a diagram of a configuration that includes two Ethernets with a variety of systems that communicate with each other by way of routing nodes.

Figure 1–1: Sample DECnet–ULTRIX Ethernet Configuration



1.2 Network Components

DECnet network components are defined as follows:

node

A node is a computer system that runs DECnet software as an interface between its operating system and the network. A node can process, send, and receive network information.

Network terminology includes three terms for describing nodes. These terms reflect your relationship to the node:

- **Local node.** Your node—the node from which you operate.
- **Remote node.** Any node other than yours.
- **Executor node.** The node where your current network management command executes.

Object

An object is a DECnet Application layer process that you can connect to over a logical link to perform general-purpose network services.

Line

A line is the physical connection between nodes. DECnet–ULTRIX software supports connections between nodes on two types of lines: DDCMP point-to-point and Ethernet. A DDCMP point-to-point line links two nodes directly; an Ethernet line links all nodes to a common media, such as a coaxial cable. Each node can access the line by means of a circuit.

DDCMP point-to-point nodes are connected to the line by DMV or DMC controllers. Ethernet nodes are connected to the line by communications controllers. (See the software product description (SPD) for a list of controllers.)

Circuit

A circuit is a logical connection that carries information between adjacent nodes and operates over the physical line. A circuit can be identical to a physical link or can be multiplexed with many other circuits on one line.

A DDCMP point-to-point circuit is the logical means for connecting your DECnet–ULTRIX node to one adjacent node on the network. Usually the adjacent node is a DECrouter in the same area. An Ethernet circuit enables you to connect to all other nodes on an Ethernet cable. Each node on an Ethernet cable is considered to be adjacent to every other node on that cable.

1.3 Network Management Tools

The DECnet–ULTRIX software offers three network management tools: the Network Control Program (**ncp**), the Event Logger (**evl**) and the **mop_mom** utility.

1.3.1 The Network Control Program (ncp)

The Network Control Program is a utility program that lets you manage the components of your network and test network performance. It also lets you display information on the condition, characteristics, and performance of network components.

All DECnet systems share a common set of commands and parameters for network management. In addition, many DECnet systems have system-specific commands and parameters. For example, DECnet–ULTRIX has system-specific commands for objects.

1.3.2 The Event Logger(evl)

The Event Logger logs network events so that you can monitor network activity. Certain **ncp** commands let you specify whether event messages are to be logged to one or all of the following sink node devices:

- Console — Any device that receives and records event messages.
- File — A user-specified file that receives event messages.
- Monitor — A user-supplied program that receives and processes events.

Some events that the Event Logger record include:

- Changes in node state.
- Passive loopback (when the executor is looping back circuit test messages).
- Line and node counter activity.

This type of information can be useful in maintaining and tuning the network, because the Event Logger can record it continuously.

See Chapter 4 for event-logging procedures.

1.3.3 The mop_mom Utility

The **mop_mom** utility lets you perform down-line loading and up-line dumping tasks. It spawns a **mop_mom** daemon that listens for down-line load and up-line dump requests on behalf of the local ULTRIX node.

When a down-line load or up-line dump request is received from a target node, the **mop_mom** utility spawns the **mop_dumpload** loader to process the load request.

For more detailed information about the **mop_mom** command, see the **mop_mom(8)** description in the *ULTRIX Command Reference* manual.

1.4 Configuration Databases

The configuration databases provide the information that a DECnet–ULTRIX node needs to function as part of a network. DECnet–ULTRIX software uses two databases: the permanent database and the volatile (running) database. You use **ncp** commands to configure network components in both databases.

During the installation of the DECnet–ULTRIX software, the installer defines the following network components in the permanent database:

- The local node and at least one other node in the network
- The local physical line
- The local circuit
- Each logging sink for the event logger

the load. (See the **syslog** description in the ULTRIX documentation for more information.)

After you issue the **trigger** command on one of the network's management hosts, the down-line load proceeds as follows (this system may also be one of your load hosts, but it is not required):

1. The management host sends a **mop remote console boot** message with the **trigger** option specified.
2. When the remote node receives this message, it multicasts a **mop request program** message.
3. The first load host that responds and the remote node, use additional **mop** messages to transfer the management host's software image into the remote node's memory. The remote node ignores other responders once the load is in progress.

1.8 Network Management Tasks

As manager of a DECnet–ULTRIX node, you have a number of key responsibilities, including:

- Configuring your node to ensure proper operation with other nodes in the network. (See Chapter 2.)
- Controlling access to the network. (See Chapter 3.)
- Monitoring Ethernet or DDCMP point-to-point operation. (See Chapter 4.)
- Testing Ethernet and DDCMP point-to-point hardware and software operation. (See Chapter 5.)

The following chapters tell you how to use **ncp** commands to perform each task. All DECnet–ULTRIX **ncp** commands discussed in these chapters are fully described in the *DECnet–ULTRIX NCP Command Reference* manual.

Chapter 2

Configuring Network Components

This chapter describes the network components and the procedures for configuring them with ncp commands. Each section describes how to specify the component and includes other procedures useful in configuring the component into the network.

All components are defined automatically during the installation of the DECnet–ULTRIX software. Use the ncp list command to display the parameter values for each component in the permanent database (see Chapter 4). You can define more nodes or modify existing ones in both the permanent and volatile databases.

Table 2-1 lists ncp commands commonly used to configure network components in the volatile and permanent databases.

Table 2–1: ncp Commands to View and Change Databases

ncp Function	Applicable Components and Devices	Volatile Database Command	Permanent Database Command
Display component data	circuit executor line logging[1] node object	show	list
Create/modify parameters for the specified component	circuit executor logging node object	set	define
Remove parameters for the specified component	circuit executor logging node object	clear	purge

[1]See Chapter 4 for information on how to specify event-logging procedures.

2.1 Configuring Nodes

Many functions the network manager performs require the identification of a specific node. During DECnet–ULTRIX installation, you define your local node name and address. You can also define node names and addresses for remote

nodes. The following sections describe node identification and the relevant node parameters for establishing an operational nodes database.

When configuring the network node database, you can use **ncp** to:

- Specify local, remote, and executor nodes by name and number
- Change the executor node identifier string
- Change the executor node state
- Reset a node's Ethernet address
- Enable proxy access on the executor node (see Chapter 3)
- Set up down-line load or up-line dump parameters

2.1.1 Specifying a Node

Specify node identification in one of the following forms:

- **Node address.** The numeric address of the node, consisting of an area number from 1 through 63, followed by a period and a number from 1 through 1,023. The second number represents the node number within the specified area. For example, a node address of 4.105 would represent node number 105 in area 4. If you are referencing a node within your area, you may omit the area number and the period separator.
- **Node name.** A unique alphanumeric character string containing 1 to 6 characters, including at least one alphabetic character.

Each node in the network must have a unique name and a unique address. Note that the node name is known only to the local node network software, while the node address is known networkwide by the routing function. Once you have specified both a node name and a node address, you can use either one whenever you need to specify a node identification in **ncp** commands.

2.1.2 Changing the Address of a Node

To change the name or address associated with a node, use the **set/define node** command.

EXAMPLE 1:

The following example changes node address 12 to node name BURGER:

```
ncp>set node 12 name burger RET
```

EXAMPLE 2:

To change the address of the executor node, execute the following command sequence:

```
set executor state off
purge node-address all
purge node node-name all
define executor address node-address
set executor state on
```

NOTE

1. If you are running Internet, Local Area Transport (LAT), or any other software that would be affected by a change in the Ethernet physical address of the executor, you must disable the software before issuing the **ncp set executor state on** and then enable it again.
2. If you have any LAT terminal lines on your system, you can use the **lcp** command to turn them off and then on again.
3. If you have Internet, you should also use the **arp -d** command to disassociate the executor with the old Ethernet physical address and to broadcast the new physical address to other Internet nodes.

2.1.3 Removing a Node

To remove a node name from the volatile database, use the **clear node** command. To remove a node from the permanent database, you must use the **purge node** command and specify the **all** keyword.

EXAMPLE 1:

The following command removes the node BOSTON from the volatile database:

```
ncp>clear node boston all RET
```

EXAMPLE 2:

The following command removes the node OHLONE from the permanent database:

```
ncp>purge node ohlone all RET
```

Some commands allow you to specify all known nodes instead of just one specific node. The **known nodes** keyword refers to all nodes that are defined in the database affected by the command you are using. For example, a **set known nodes** command affects all nodes currently defined in the volatile database but leaves node definitions in the permanent database unchanged.

2.1.4 Changing the Executor Node Identifier String

During DECnet–ULTRIX installation, you can provide a string of information that appears on the screen whenever you display executor node information.

To modify this string, use the **identification** parameter with the **set executor** or **define executor** command in the following format:

ncp set executor identification *id-string*

When you issue this **ncp** command at the shell prompt, you must follow these conventions: for the shell, enclose any string containing blanks or tabs in double quotation marks; for **ncp**, use single quotes.

EXAMPLE 1:

The following example shows the **ncp define executor identification** command executed from the shell:

```
% ncp define executor identification 'DECnet—ULTRIX BOSTON V4.0' RET
```

EXAMPLE 2:

This example shows the same command executed from within **ncp**:

```
ncp>define executor identification "DECnet—ULTRIX BOSTON V4.0" RET
```

EXAMPLE 3:

To quote a word within the identification string, enclose the word within two sets of double quotation marks. For example:

```
% ncp define executor identification 'DECnet—ULTRIX ""BOSTON"" V4.0' RET
```

2.1.5 Changing the Executor Node State

When your DECnet–ULTRIX system is installed, the executor node comes up in the **on** state and is available for use. If you do not want your system to come up with the executor node on, you must edit the **/etc/rc.local** file after installation and before rebooting to remove the **ncp set executor state on** command.

NOTE

If you have Local Area Transport (LAT) terminal lines on your system, you should not delete the **ncp set executor state on** command from the **/etc/rc.local** file unless you start TCP/IP with the **ifconfig** command. Either DECnet or TCP/IP must be running for LAT to function.

Once your system is running, you can use the following **set executor** command to change the state:

set executor state { **on** | **restricted** | **shut** | **off** }

where

on	specifies normal operation. Allows logical links to and from the node.
restricted	specifies limited operation. Allows no new inbound links.
shut	specifies orderly shutdown. Allows no new logical links to or from the node, but does not terminate existing links. After all links have disconnected, the executor goes into the **off** state, thereby shutting down the network at the local node.
off	specifies immediate shutdown. Immediately terminates all network activity without allowing active links to disconnect in an orderly manner. All active links are aborted and active tasks are notified of network shutdown.

2.1.6 Resetting a Node's Ethernet Address

Each DECnet node on the Ethernet has a unique node address that allows it to communicate with any other node on the same Ethernet. The Ethernet address on your Ethernet controller is reset by the DECnet software whenever the **ncp set executor state on** command is executed.

NOTE

This usually occurs during system installation. However, it can occur later if you redefine your node address.

2.1.7 Down-Line Loading or Up-Line Dumping a Node

One way to set up the parameters for down-line loading from a DECnet–ULTRIX node is to use the following command format:

ncp set node *node-id* **load file** *filename* RET

One way to set up the parameters for up-line dumping to a DECnet–ULTRIX node is to use the following command format:

ncp set node *node-id* **dump file** *filename* RET

For more information about using **ncp set node** or **define node** commands, see the *DECnet–ULTRIX NCP Command Reference* manual.

2.2 Configuring Network Objects

To configure a network object, use the following:

- Specify an object by name or number.
- Define an object file name.

2.2.1 Specifying an Object

All objects are identified by an object name and a number from 0 through 255, depending upon the type of object.

- **Zero objects** are user-defined processes for special-purpose applications. All objects assigned to object number 0 are identified by an object name (1 to 16 alphanumeric characters). You must use this object name when issuing a logical link connect request.

  An undefined object is a zero object that is not defined in your object list.

- **Nonzero objects** usually are DECnet-supplied processes, such as File Access Listener (**fal**) and Network Management Listener (**nml**), that provide a specific, cross-system network service. However, you can also supply user-written tasks for known network services. Each object is identified by an object number ranging from 1 through 255. DECnet-supplied objects that perform the same function on different systems are all assigned the same object number, even if their object names are different. For example, the DECnet–ULTRIX network terminal handler (**dterm**) and the DECnet–VAX terminal handler (REMACP) are both assigned object number 23. Numbers from 1 through 127 are reserved for Digital-supplied services; numbers from 128 through 255 are available for customer-supplied services.

2.2.2 Defining an Object File Name

Each DECnet-supplied object invokes an executable file that is defined for that object (use the **ncp show known objects** command to display file names defined for objects on your system). The DECnet-ULTRIX kit supplies a zero object named DEFAULT that does not have an executable file associated with it. You can use the **ncp set/define object** command to define a DEFAULT file name or you can leave the DEFAULT object file undefined, depending upon what action you wish to take when an undefined object name (that is, a zero object that is not defined in your object list) is received on a connect request:

- If DEFAULT has a defined file name, any zero object that is not defined in your object list executes the file defined for DEFAULT.
- If no file name is defined for DEFAULT and an undefined zero object is received on a connection request, the system software searches for the supplied object name according to the search path for the default user account associated with DEFAULT. If the software finds a file that matches the supplied object name, it executes that file. If it does not find a matching file name, it rejects the connection request.

To define the zero object, enter:

```
ncp>set object RET
```

or

```
ncp>define object RET
```

For information about the **ncp set object** or **define object** commands, see the *DECnet–ULTRIX NCP Command Reference* manual.

2.3 Configuring Lines

To configure a line do the following:

- Specify a line by device name and number.
- Change a line's state.

2.3.1 Specifying a Line

A line ID has the following format:

dev-c

where

dev is a DECnet–ULTRIX line device name. The following table contains the DECnet device names with the equivalent Internet names:

DECnet Device Names	Equivalent Internet Device Names
una	de
qna	xna
dmc	dmc
dmv	dmv
sva	se
bnt	ni

c is a number from 0 through 65,535 that designates the device's hardware controller.

NOTE

The **ncp** commands **list known lines** and **list known circuits** display only lines or circuits configured on the ULTRIX system and running on DECnet. If one of these commands does not display the circuit or line you are looking for, enter a **show** or **list** command for that entity.

The following command displays the circuit una-0:

```
ncp>list circuit una-0 RET
```

2.3.2 Changing a Line's State

When you install DECnet–ULTRIX, the line on which you choose to run DECnet turns on and is available for use when the system comes up.

DECnet and Internet can share the same Ethernet line. You can control DECnet's access to the Ethernet by setting the circuit's **state** with the **ncp set** or **define circuit** command.

EXAMPLE 1:

This command turns the DECnet network on:

```
ncp> set circuit una-0 state on RET
```

With an Ethernet line you can leave the line state on even when you are not using DECnet.

With a DDCMP point-to-point line you must switch between running DECnet and Internet. When you install DECnet–ULTRIX, DECnet is on. To switch to Internet, you must turn off DECnet, and then turn on Internet. The **ncp set exec state** command turns off DECnet, and **ifconfig(8)** turns on Internet.

EXAMPLE 2:

This command turns off DECnet:

```
% ncp set exec state off RET
% /etc/ifconfig dmv-0 boston 1.16 netmask 255.0 RET
```

To switch back to DECnet, turn off Internet and reset the DECnet executor state to **on**.

EXAMPLE 3:

This command turns off Internet and resets the DECnet executor state to **on**:

```
% /etc/ifconfig dmv-0 down RET
% ncp set exec state on RET
```

See the ULTRIX network management documentation for more information about turning Internet on and off.

For more information about configuring lines, see the *DECnet–ULTRIX NCP Command Reference* manual.

2.4 Configuring Circuits

To configure a circuit do the following:

- Specify a circuit by device name and number.
- Change a circuit's state.

2.4.1 Specifying a Circuit

To use **ncp** commands to modify or display circuit parameters, you must specify an individual circuit by using its circuit ID in the following form:

dev-c

where

dev	is one of the following DECnet–ULTRIX circuit names: una, qna, dmv, dmc, sva, or bnt.
c	is a number from 0 through 65,535 that designates the device's hardware controller.

2.4.2 Changing a Circuit's State

If you do not want the circuit turned on the next time your system comes up, you can issue an **ncp define circuit** *circuit-id* **state off** command to set the circuit off in the permanent database.

While your system is running, you can use the **set circuit** command to change the circuit state. Changing the circuit state controls DECnet access to the line without affecting the line state.

Chapter 3

Controlling Network Access

This chapter explains how to use access-control information for commands you want to execute on a remote node. It also tells you how to set up proxy for incoming connection requests.

3.1 Access Restrictions on Remote Nodes

Before you issue commands to be executed at a remote node, you may have to supply access-control information. Depending on the type of access restrictions set up by the system manager of the remote node, you can gain access to a remote node from a DECnet–ULTRIX node in the following ways:

- Append access-control information to the node identification in the **ncp** command string.
- Include access-control information in an alias definition for the node.
- Use proxy verification on the remote node.

For procedures on how to supply access-control information, see the *DECnet–ULTRIX Use* manual.

3.2 Appending Access-Control Information

If you append access-control information to the node identification on any **ncp** command line, the remote node verifies your log-in name and password. It does this by checking the log-in name and password against its password file (user authorization file). If the password file contains a matching entry, the remote node executes the command; if not, the node rejects the command.

Use the following format to append access-control information to the node name:

node-id **user** *login-name* [*password*]

You can also use the ULTRIX format:

node-id/login-name[*/password*]

where

node-id	is the name or address of the remote node.
login-name	is a string of up to 39 alphanumeric characters that identifies a user on the remote node. The log-in name for an ULTRIX user is the account name.

password is a string of up to 39 alphanumeric characters identifying the remote user's log-in name. The remote system uses this string to verify the identity of the user.

If you do not want the password to echo, type a question mark (?) in place of the *password* string.

If you are using the ULTRIX format, type a backslash and a question mark (?) in place of *password*. For example:

```
ncp>tell boston/jill/ \? set exec gateway access enable RET
```

After you press RET the system prompts you for your password and does not echo it as you type.

If you choose not to enter a password, press RET at the password prompt.

3.3 Defining an Alias

As a shortcut to typing the remote node identification and the required access-control information, you can specify an alias node name. An alias node name is an alphanumeric string of one or more characters that you type in place of a node identification and access-control information. You can define alias node names by creating a **.nodes** file in your home directory. Use the following format for entries in this file:

alias=node-id[*/login-name*[*/password*]]

NOTE

Do not use spaces or tab characters in any of the fields.

EXAMPLE 1:

This example shows three ways to add **.nodes** file entries:

```
b=boston
w=boston/root/xyzkoroijt
payroll=boston/hart/yxkowilk
```

EXAMPLE 2:

This example shows how the alias **w** represents the string **boston/root/xyzkoroijt** in a **tell** command string.

```
% ncp tell w set execution gateway access enable RET
```

CAUTION

To prevent unauthorized access to your passwords, set up the protections on the **.nodes** file so that only the owner can read the file or write to it.

3.4 Using Proxy Verification

Proxy verification lets you execute commands on a remote node without supplying access-control information. It is more secure than sending your password over the network. Although DECnet–ULTRIX software supports proxy verification, not all DECnet systems do. Contact the manager of any non-ULTRIX system to find out if it supports proxy verification.

Before you can use proxy verification, the system manager for the remote node must set up a proxy account for you. If you are going to have access to more than one proxy account from the same node and log-in name, indicate which proxy account should be the default.

To use your default account, enter the command without any access-control information.

EXAMPLE:

This example shows how to use the **tell** command to enter a command without access-control information.

```
ncp>tell navaho set exec gateway access enable RET
```

To use an account other than the default, append the account log-in name to the node identification.

EXAMPLE:

This example shows how to use the **tell** command to append the log-in name to the node identification.

```
ncp>tell boston/rudi set exec gateway access enable RET
```

3.5 Controlling Proxy Access on a Node

The following sections describe how to set up and manage a proxy file, control proxy access to your node, and control outbound proxy requests.

3.5.1 Setting Up a Proxy File

To set up a local proxy file, make an entry in the file **/etc/dnet_proxy** for each user to whom you want to permit proxy access. To set up more than one proxy account for a user, list the entry for the user's default account above any other entries for that user. Use the following format for the entries:

source-node-name::*source-login-name destination-login-name object* [*#comment*]

where

source-node-name	is the name of the remote node from which you are allowing proxy access on your node.
source-login-name	is the remote user for whom you are allowing proxy access.
destination-login-name	is the account on your node to which you are allowing proxy access.
object	is a list of objects, separated by commas, for which the proxy entry is valid. (Optional.)

#comment	is an alphanumeric string of one or more characters for notes about the proxy account.

You can use a wildcard character in place of the source node name, source log-in name, or destination log-in name to indicate that any entry is valid for that field.

The following examples show a proxy file and explain each of its four entries.

EXAMPLE 1:

This example shows a proxy file with four entries:

```
*::jones jones #Jane Jones
school::* newuser #account for new students
navaho::* guest fal
boston::* * #Accounts for node BOSTON users
```

EXAMPLE 2:

This entry means that remote user **jones** has access from any node to account **Jones** on your node.

```
*::jones jones #Jane Jones
```

EXAMPLE 3:

This entry indicates that any user on node SCHOOL has access to account **newuser** on your node.

```
school::* newuser #account for new students
```

EXAMPLE 4:

This entry indicates that any user on NAVAHO can access object **fal** through the account **guest**.

```
navaho::* guest fal
```

EXAMPLE 5:

This entry indicates that any user on node BOSTON can access an account with the same name on your node.

```
boston::*     * #Accounts for node BOSTON users
```

CAUTION

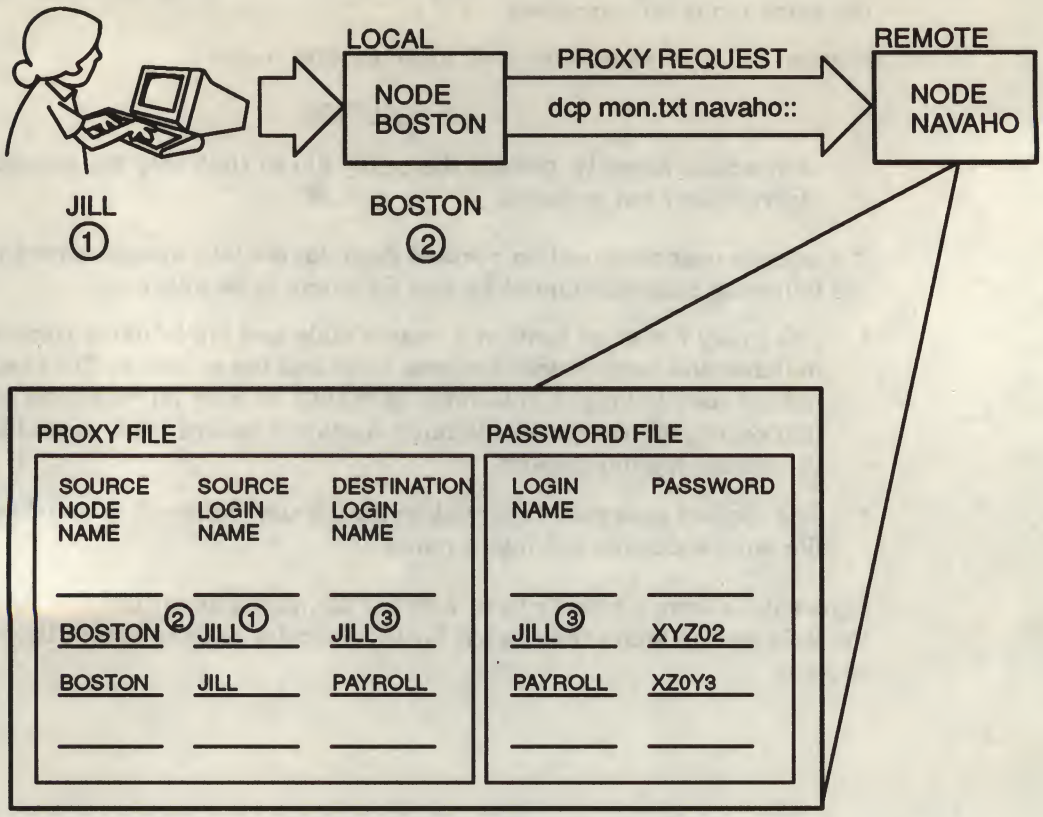
For added security, protect the proxy file so that only the superuser (privileged) can access it.

If a remote user's connection request does not contain access-control information, the following conditions must be met for proxy to be approved:

- The proxy file must contain a source node and log-in name combination that matches the remote user's source node and log-in name. For example, if a remote user is logged in to node BOSTON as user **jill** when she executes an **ncp** command, the proxy file must contain a source node name BOSTON with the source log-in name **jill**.
- The system password file must contain a user name that matches the proxy file entry's destination log-in name.

Figure 3-1 shows a remote user, a proxy file, and a password file. Lines connect the information that must match for the executor node to accept the remote user's request.

Figure 3–1: Proxy Request Without Access-Control Information

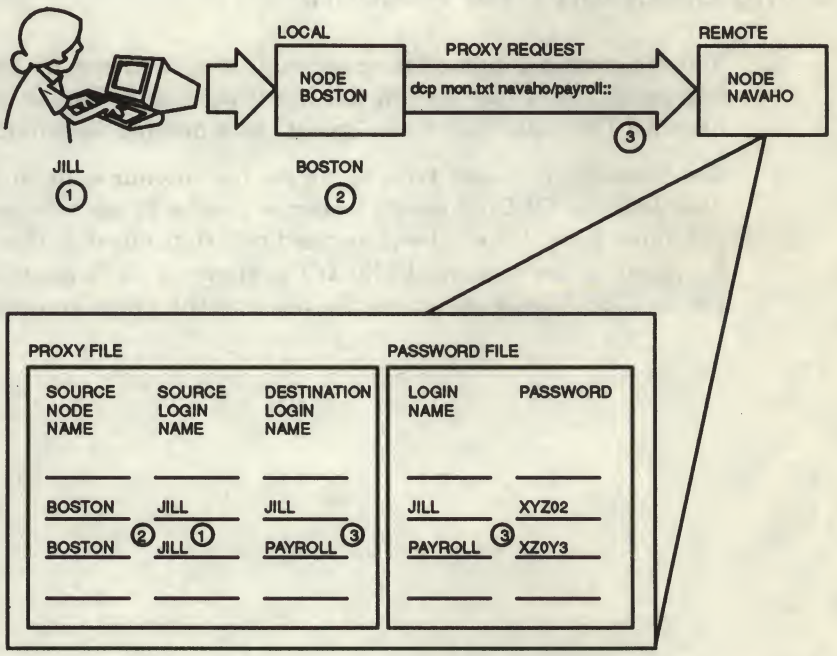


If a remote user appends a user name to the node identification, the following must hold true for proxy to be approved:

- The proxy file must contain an entry in which the source node and log-in name combination matches the remote user's source node and source log-in name.
- This proxy file entry must also contain a destination log-in name that matches the user name that the user appended to the node identification.
- The system password file must contain a user name that matches the proxy file's destination log-in name.

Figure 3–2 shows the required matches:

Figure 3–2: Proxy Request With Access-Control Information



If any one of these requirements is not met or **incoming proxy** is disabled, the system can approve access only if the user name appended to the node identification is listed in the system password file with a null password.

3.5.2 Enabling or Disabling Incoming Proxy

The executor parameter **incoming proxy** lets you control connect requests to your node. If proxy is disabled, the system will reject an incoming connection request. This parameter is enabled by default. To turn off proxy access to your node, set this parameter to **disable**.

EXAMPLE:

This example shows how to use the **set executor** command to turn off proxy access:

```
ncp>set executor incoming proxy disable RET
```

The default for the **incoming proxy** parameter is **enable**.

3.5.3 Enabling or Disabling Outgoing Proxy

You can prevent outbound connection requests from leaving your node by setting the executor's **outgoing proxy** parameter to **disable**.

EXAMPLE:

This example shows how to use the **set executor** command to disable outbound proxy requests.

```
ncp>set executor outgoing proxy disable RET
```

The default for this parameter is **enable**. For more information, see the **set executor** or **define executor** command in the *DECnet–ULTRIX NCP Command Reference* manual.

3.6 Using a Default User Account

You can define a default user account, and then specify the default user for the requested object (service) in the object database. Unless you redefine the default user, all DECnet objects use "guest" as a default user name.

For example, if a user tries to copy a file to your node and **incoming proxy** is disabled, the DECnet object spawner checks to see whether the object **fal** has a default user. When the spawner finds that guest is the default user, it looks for guest in the password file. If the spawner finds guest in the password file, it verifies access and **fal** copies the file into the guest account.

Chapter 4

Monitoring Network Activity

DECnet provides several means of monitoring network activity from a DECnet node. For example, you can:

- Display information about network components by using the **ncp show** or **list** commands.
- Use **ncp** to request the Event Logger (**evl**) to log network events.
- Measure network performance by evaluating the DECnet counters for circuits, lines, and nodes.
- Monitor access to your local node by logging certain object activities to a log file.

All of these facilities are described in this chapter. See the *DECnet–ULTRIX NCP Command Reference* manual for details on the **ncp** commands.

4.1 Using ncp Display Commands

The **ncp show** and **list** commands display information about network components. The **list** command displays information from the permanent database, while the **show** command displays component information from the volatile database.

The components for which you can display information include:

- Network components:
  - Circuits
  - Lines
  - Nodes
  - Objects
- Event Logger components:
  - Console
  - File
  - Monitor program

You can display information for a specific component or for all known components of the specified type. For example, if you want to see which nodes are currently reachable, you can issue a **show known nodes** command. You can also display a subcategory of **known** for nodes by specifying **show active nodes**, which displays information about known nodes that are currently turned on.

Depending on the component you specify in a **show** command, you can select from the following displays:

characteristics	displays static information about the component, such as the parameters defined for that component. This information is kept in either the volatile or permanent database.
counters	provides counter information for circuits, lines, and nodes. (See Section 4.3 for a discussion of counters and a sample display.)
events	displays information about events currently being logged. Valid for **show** or **list logging** only.
status	shows information that usually reflects network activity for the running network. Depending on the component, this information can include the local node and its operational state, reachable and unreachable nodes, and circuits and lines and their operational states.
summary	includes only the most useful information, which is usually an abbreviated list of information provided for both the **characteristics** and **status** display types. (Default.)

The following examples demonstrate some **ncp show** commands and their resulting displays.

EXAMPLE 1:

To display line characteristics, enter:

```
ncp>show line una-0 characteristics RET
Line Volatile Characteristics as of Wed Jun 19 16:38:06 EDT 1990
Line  =  UNA-0
Controller              = Normal
Protocol                = Ethernet
Hardware address        = aa-00-03-00-00-99
```

EXAMPLE 2:

To display characteristics of a specific object, enter:

```
ncp>show object nml char RET
Object Volatile Characteristics as of Thu Jun 14 10:07:11 EST 1990
Object = nml
Number                  = 19
File                    = /etc/nml
Default User            = guest
Type                    = Sequenced Packet
Accept                  = Deferred
```

4.2 Using Event Logging

After the DECnet–ULTRIX software is installed, **evl** begins recording network events, such as changes in node state, loopback test messages, and line and node counter activity.

By default, all events for all components known to the local node are logged at the executor console. However, you can use **ncp set logging** or **define logging** commands to specify the event classes or types you want to log. You can also inhibit or enable logging to the console, a file, or user program.

The following sections describe how to customize the Event Logger.

4.2.1 Displaying Event Logger Status

Before you can begin customizing the Event Logger, view the status of the existing components.

EXAMPLE:

To display all existing logger components, enter:

```
ncp>show known logging RET

Known Logging Volatile Summary as of Thu Jun 14 10:10:51 EST 1990

Logging = console

State                           = On
Name                            = /dev/console
Sink node                       = 7.3 (ATLANT), events =
                                  0.0-6
                                  2.0-1
                                  3.2
                                  4.3-10,18

Logging = file

State                           = Off
Name                            = /usr/adm/eventlog

Logging = monitor

State                           = On
Name                            = /etc/evl
```

4.2.2 Specifying an Event Class and Type

Events are defined by class and type. You can specify class and type of events to be logged by using the following event list format with the **events** parameter in a **set** or **define logging** command:

class.type[,*type,type,...,type*]

where

class	identifies the DNA layer or system-specific resource to which the event pertains.
type	identifies the particular event within the event class. The *type* variable can be a single number or a range of numbers.

When providing an event list for the **events** parameter, you can specify only one class, but you can specify multiple event types within a class. For example, you can specify a single event type, a range of types, a combination of these, or a wildcard character. The following sample event list illustrates the different formats:

Event List	Meaning
2.0	Specifies event class 2, type 0.
4.15-18	Specifies event class 4, types 15 through 18.
4.3,8-10,18	Specifies event class 4, types 3, 8 through 10, and 18. Note that types must be specified in ascending order.
0	Specifies all events in class 0.

See the *DECnet–ULTRIX NCP Command Reference* manual for a list of all events (by class and type) that DECnet–ULTRIX can log. To determine the events logged by a remote node, see the DECnet documentation for the remote system. To specify logging of all network event classes and types, use the **known events** parameter.

4.2.3 Event Sources

Event sources are qualifiers for events. If no source is specified, logging events for all sources are affected by the command. When you specify a source for events, only events generated by that source are affected. Sources you can specify for DECnet–ULTRIX events are **circuit**, **line**, or **node**.

EXAMPLE:

To monitor network activity over line una-0, connected to the local node, use the following command:

```
ncp>set logging console known events line una-0 RET
```

This command causes all events that pertain to line una-0 to be logged at the console by the Event Logger.

4.2.3.1 Specifying Logging Component Names

When you initially specify logging components to which event data is to be logged, you can identify them by using the **name** parameter in the **set logging** command. Use the **clear logging name** command to clear the name of the logging component and cause events to be sent to the default device for that logging component. The default logging names are as follows:

- For the console logging component: **/dev/console**
- For the file logging component: **/usr/adm/eventlog**
- For the monitor logging component: **evl**

4.2.3.2 Logging Sinks

You can use a sink qualifier with any of the logging components. Use the **sink** parameter to indicate the location of the logging component. The sink can be the executor node or a remote node.

EXAMPLE:

This command routes all events to the console logging component on node NAVAHO.

```
ncp>set logging console known events sink node navaho RET
```

If you do not specify a sink qualifier, the local node is the default component location. If the sink node is unreachable when the event occurs, the event information is discarded.

4.2.3.3 Event-Logging Component States

You can inhibit or enable logging to the console, a file, or user program by setting the **state** parameter of that component to **on** or **off**. The **state** parameter causes all events destined for the logging component to be discarded. The **on** state enables logging on the specified logging component.

EXAMPLE:

This command disables logging to the system operator's terminal on the local node.

```
ncp>set logging console state off RET
```

NOTE

This does not affect logging to any other logging component whose state may be **on**.

4.2.4 Monitoring Local and Remote Nodes

You can use event-logging commands to monitor local and remote nodes in your network.

EXAMPLES:

The sequence of **ncp** commands, shown in the following examples, sets up event logging for a local node called TOM and for two remote nodes called DICK and HARRY.

EXAMPLE 1:

In response to this command, TOM's system console displays all known events that occur locally:

```
ncp>set logging console known events state on RET
```

EXAMPLE 2:

In response to the following four commands, remote nodes DICK and HARRY each log all known events locally at the system console and remotely at node TOM's system console.

```
ncp>set logging console known events state on RET
ncp>tell dick set logging console known events state on RET
ncp>tell dick set logging console sink node tom known events RET
ncp>tell harry set logging console known events state on RET
ncp>tell harry set logging console sink node tom known events RET
```

4.3 Reading Counters

DECnet–ULTRIX software maintains certain statistics, called counters, for circuits, lines, and nodes (including the executor). All counters are listed and briefly described in the *DECnet–ULTRIX NCP Command Reference* manual.

Counters are maintained on the node presently designated as the executor and can include such information as the number of data packets sent, received, or lost over a line; the number of connect messages sent or received; system buffer allocation failures; and routing packet information. Counter statistics are useful alone or when read in conjunction with logging information to measure and evaluate the performance and throughput of your network configuration.

You can display counters and periodically reset them to zero using the **show** and **zero** commands.

4.3.1 Displaying Counters

You can use the **ncp show** command described in the following examples to display counters for circuits, lines, and nodes.

EXAMPLE:

This example shows a sample command and the resulting display:

```
ncp>show line una-0 counters RET

Line Volatile Counters as of Wed Jun 19 16:40:22 EDT 1990
Line  =  UNA-0
  12770 Seconds since last zeroed
5703244 Bytes received
3695497 Bytes sent
2498504 Multicast bytes received
  87029 Data blocks received
  65294 Data blocks sent
  23688 Multicast blocks received
   1477 Blocks sent, initially deferred
     91 Blocks sent, single collision
     83 Blocks sent, multiple collisions
      0 Send failure
      0 Collision detect check failure
      3 Receive failure, including:
          Framing error
          Frame too long
   1996 Unrecognized frame destination
      0 Data overrun
      0 System buffer unavailable
      0 User buffer unavailable
```

The Ethernet header size is incorrectly included in the count for Bytes received and Multicast Bytes received on DESVAs and DEQNAs.

Some counters can be qualified by information that indicates the condition(s) that contributed to an error.

Some network logging events relate to the network counters; for example, event 0.5 logs node counter values before they are zeroed. See the *DECnet–ULTRIX NCP Command Reference* manual for a complete description of events that can be logged.

4.3.2 Zeroing Counters

When the network is running, you can use the **ncp zero** command to reset any of the network counters to zero. It is wise to zero counters periodically so that they do not exceed (overflow) their maximum count. Counters that have overflowed display a "greater than" sign (>) in front of their maximum count (for example, >65534 means that the maximum count of 65,534 has been exceeded).

Each component has a special counter that indicates the number of seconds that have elapsed since the counters for that component were last zeroed. The software increments this counter every second and zeros it when other counters for the component are zeroed.

4.4 Monitoring Access with Object Log Files

DECnet–ULTRIX logs specific information that can be used by the network manager to monitor user access of the local system. Four programs use the ULTRIX **syslog** subroutine to log such information: the DECnet object spawner, the File Access Listener (**fal**) and the remote terminal access server programs (**dlogin** and **dterm**). For more information on **syslog**, see **syslog(3)** in the ULTRIX reference documentation.

The user-access information is logged to file **/usr/spool/mqueue/syslog** by default. Table 4-1 lists the DECnet programs that log this information and notes the priority level and type of information logged for each.

Table 4–1: DECnet–ULTRIX Log Files

Program	Priority Level	Type of Information Logged
dnet_spawner	LOG_DEBUG(9)	All connects to and exits from DECnet objects. Also logs connect requests that are rejected by the DECnet spawner (for example, requests with bad access-control information).
fal	LOG_INFO(8)	All attempts at file access.
dlogin	LOG_INFO(8)	All remote terminal connects and disconnects. Supports DECnet heterogeneous command terminal specification (CTERM) for DECnet–ULTRIX and VMS V4.0 and later.
dterm	LOG_INFO(8)	All remote terminal connects and disconnects. Uses TOPS-20 homogeneous command terminal protocol.

You can set up the **syslog** configuration file (**/etc/syslog.conf**) to have this information logged to any files that you want. You can filter out logging of priority 9 data by setting the file priority to 8, or you can bypass logging of all of these programs to a file by setting the file priority to 7 or lower.

Chapter 5

Testing the Network

This chapter describes loopback and DECnet Test Sender /DECnet Test Receiver (**dts/dtr**) tests and explains how to use them.

5.1 Loopback and dts/dtr Tests

Several kinds of tests help you determine whether the network is operating properly. Among them are loopback and **dts/dtr** tests, which you can run on your nodes.

- **Loopback Tests.** Loopback tests (node level and circuit level) let you exercise network software and hardware by first sending data through various network components and then returning that data to its source for data comparison. After you have started DECnet–ULTRIX software, you can use **ncp loop** commands to test network performance.
- **dts/dtr Tests.** The **dtr** program functions as a slave to **dts** and must exist as defined object 63 at the remote node. The **dts** program initiates each test by issuing a connection request to **dtr**. Parameter information pertinent to the type of test requested is passed by **dts** to **dtr** in the optional data of the connection request. The **dts** user interface enables the user to issue commands with options to customize the test to be performed. Parameters are available to regulate such variables as message length, test duration, and type of data used.

5.2 Node-Level Loopback Tests

Node-level loopback tests check the logical link capabilities of a node by exchanging test data between DECnet tasks in two different nodes or between DECnet tasks in the same node. Two types of node-level tests allow you to test different layers of DECnet–ULTRIX software:

- **Local-to-local loopback tests** verify local-node network operation.
- **Local-to-remote loopback tests** verify network operation between the local node and a remote node.

Use the node-level tests first. Then, if further testing is desired, use the circuit-level tests.

To perform these loopback tests, use the **ncp loop node** command. The **loop node** operation uses the two cooperating tasks Looper and Loopback Mirror (**mlr**). When you issue the **loop node** command, you can specify the following information:

- The **with** parameter, which specifies the type of binary information used to perform the test. The possible values are **ones**, **zeros**, and **mixed**. The value **mixed**, a combination of ones and zeros, is the default.
- The **count** parameter, which specifies the number of data blocks to be sent during the test. It is a number from 1 (default) through 65,535.
- The **length** parameter, which specifies the length in bytes of each block to be looped. This value must be a decimal integer from 1 through *n*, where *n* must be less than the smaller of either the local looper buffer size or the remote mirror buffer size. The default is 40 bytes.

If a test completes successfully, **ncp** prompts you for the next command. If a looped message returns with an error, the test stops and **ncp** prints a message specifying the reason for the failure. If a connection was attempted, **ncp** provides a count of the messages that were not returned.

EXAMPLE:

In the following test, a network manager attempts to send a message 10 blocks long to node BOSTON. The result is that the message is not looped because node BOSTON is unreachable.

```
ncp>loop node boston count 10 RET
ncp - listener response: Mirror connect failed, Node unreachable
Unlooped count = 10
ncp>
```

See the *DECnet–ULTRIX NCP Command Reference* manual for a complete list of error messages.

5.2.1 Local-to-Local Loopback Test

The local-to-local loopback test evaluates the local node's DECnet software using an internal logical link path; no physical device is used.

EXAMPLE:

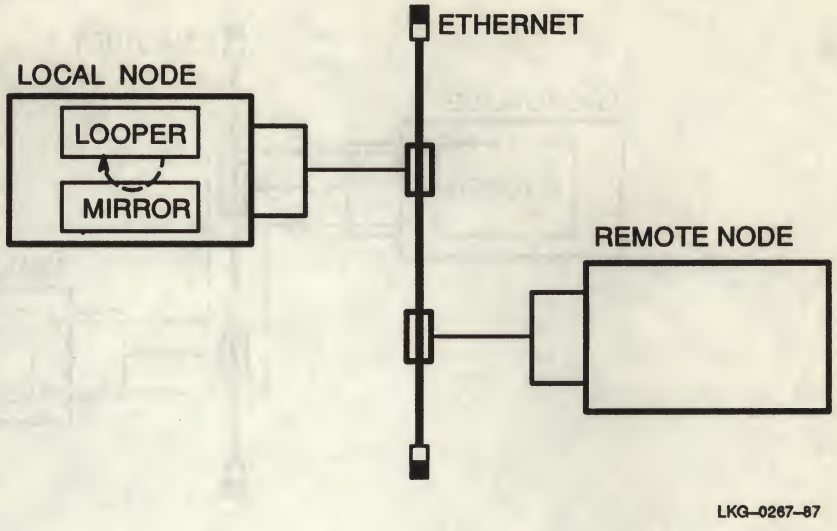
For this test, you simply issue the **ncp loop executor** command:

```
ncp>loop executor count 10 RET
```

The local-to-local loopback test verifies operation of the local Network Application layer, Session Control layer, End Communications layer, and part of the Routing layer. A failure of this test indicates a problem with the local node software, such as the network being turned off or access control to the mirror not being properly established. If the local-to-local loopback test succeeds, you should perform a local-to-remote loopback test. If the local-to-remote loopback test succeeds and the local-to-local test fails, try the circuit-level tests to determine if the hardware is at fault.

Figure 5-1 illustrates a local loopback test.

Figure 5–1: Local-to-Local Loopback Test



5.2.2 Local-to-Remote Loopback Test

This test verifies operation of all levels of network software on the local and remote nodes you are testing. When you use this command, you must identify the node to which you want to loop test messages. This node must be reachable over circuits that are in the **on** state.

Figure 5-2 illustrates a local-to-remote loopback test.

Before doing this test, use the **ncp show circuit summary** command to see whether the test circuit's state is **on**. If not, issue an **ncp set circuit state on** command for that circuit. Then use the **ncp loop node** command to initiate the loopback test.

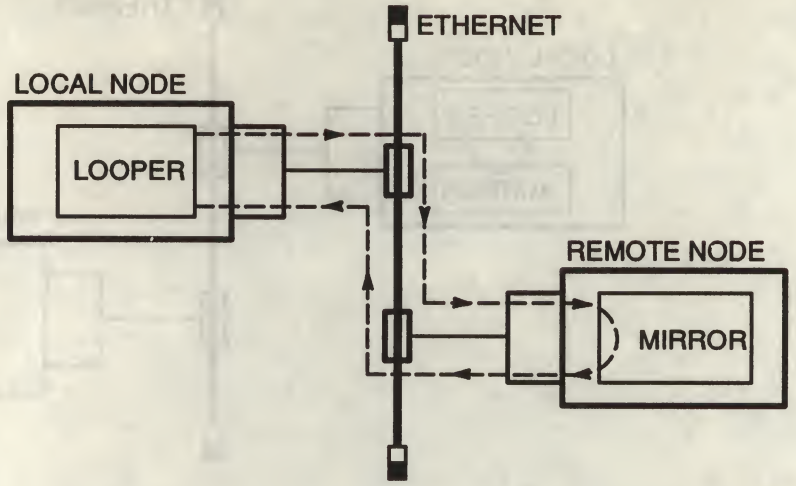
EXAMPLE:

The following command tests DECnet software on both the local node and on remote node TOLEDO:

```
ncp>loop node toledo count 10 RET
```

If the previous local-to-local tests were successful and this test fails, a problem exists with either the remote node or the DECnet circuit.

Figure 5–2: Local-to-Remote Loopback Test



5.3 Circuit-Level Loopback Tests

Circuit-level loopback tests check a DECnet circuit by looping test data between DECnet tasks in two different nodes or between DECnet tasks in the local node. These tests use a low-level data link interface rather than the logical links used by the node-level tests. There are two types of circuit-level tests:

- **Software loopback tests** loop the data through an adjacent node on the circuit to determine whether the circuit is operational up to the adjacent node's circuit unit and controller.
- **Controller loopback tests** loop the data through the device on the circuit in loopback mode to determine whether the controller works properly.

To perform these loopback tests, use the **ncp loop circuit** command.

When you run DECnet software on a diskless client, do not set the controller loopback mode with the **ncp** command **set line** *dev*-n **controller loopback**. Setting the controller to loopback mode causes the client to lose contact with its server and, as a result, to lose access to its file system.

The **ncp loop circuit** commands may not work when you issue them from a VMS node to an ULTRIX node on a DDCMP point-to-point line or an Ethernet cluster; they will work from an ULTRIX node to a VMS node.

When you issue the **loop circuit** command, you can specify the following information:

- The **with** parameter, which specifies the type of binary information used for the test. The possible values are **ones**, **zeros**, and **mixed**. The value **mixed**, a combination of ones and zeros, is the default.
- The **count** parameter, which specifies the number of data blocks to be sent during the test. It is a number from 1 (default) through 65,535.

- The **length** parameter, which specifies the length (in bytes) of each block to be looped. This value must be a decimal integer in the range of 1 through *n*, where *n* must be less than the smaller of either the local looper buffer size or the remote mirror buffer size. On the Ethernet, the allowable length is from 1 byte to the maximum length of the data pattern, which varies according to the level of assistance. The default is 40 bytes.

Level of Assistance	Maximum Length
No assistance	1486 bytes
Transmit or receive assistance	1478 bytes
Full assistance	1470 bytes

If a test completes successfully, **ncp** prompts you for the next command. If a looped message returns with an error, the test stops and **ncp** prints a message specifying the reason for the failure. If a connection was attempted, **ncp** provides a count of the messages that were not returned.

EXAMPLE:

This test attempts to send 10 messages. The first four messages are sent successfully, and an error occurs on the fifth, causing the test to halt.

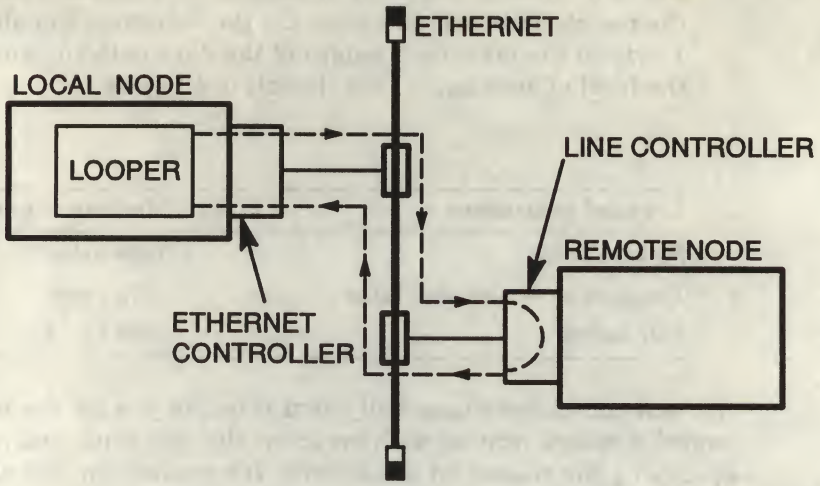
```
ncp>loop circuit una-0 count 10 RET
ncp - listener response: Line communication error
Unlooped count = 6
ncp>
```

5.3.1 Software Loopback Test

This test verifies that the circuit is operational up to the unit and controller on the adjacent node. This type of test uses DECnet–ULTRIX software to loop data through the circuit-to-circuit service software in the adjacent node and back to the local node. You can specify optional parameters for assistance in testing a remote node. Figure 5-3 illustrates a software loopback test.

Before doing this test, use the **ncp show line characteristics** command to see whether the line controller is in **normal** mode; if it is not, issue an **ncp set line controller normal** command for the line. Then issue an **ncp loop circuit** command to test the circuit between your node and an adjacent node. If this test fails, try a controller loopback test to see whether the controller is functional.

Figure 5–3: Circuit-Level Software Loopback Test



5.3.1.1 Software Loopback Testing Over Ethernet Devices

There are two ways of performing the software loopback test on the Ethernet:

- Loop to any random node on the Ethernet.
- Loop to a specific node on the Ethernet.

Before performing either test, issue an **ncp show circuit summary** command to see whether the circuit is in the **on** state; if it is not, issue an **ncp set circuit state on** command for that circuit.

Random Node Loopback Testing: You can send a test message to all nodes on the Ethernet by way of a multicast message.

EXAMPLE 1:

This command sends a test message to all nodes on the Ethernet. If the **count** parameter is greater than 1, the first node to respond to the multicast message will loop the remaining test messages.

```
ncp>loop circuit una-0 count 50 RET
```

EXAMPLE 2:

A return message is displayed that indicates the responding node's physical address:

```
Loop succeeded
Physical Address = AA-00-04-00-23-04
```

Specific Node Loopback Testing: You can specify a remote node on the circuit that you want to test by using either its node name or physical address.

NOTE

Nodes on Ethernet circuits are identified by unique Ethernet addresses. If the node is running DECnet, this physical address is the address that DECnet has created using the DECnet node address. If the node is not running DECnet, the physical address is the default hardware address of the node.

To perform the test using a specific remote node, specify the **physical address** parameter along with its value.

EXAMPLE:

This command loops messages through the local device una-0 to the remote node having physical address AA-00-03-00-FF-08:

```
ncp>loop circuit una-0 physical address aa-00-03-00-ff-08 RET
```

5.3.1.2 Ethernet Loopback Assistance

DECnet supports the use of an assistant node to aid you in interrogating a remote node. To use this assistant feature, specify either the **assistant physical address** parameter or the **assistant node** parameter as an additional parameter to the **ncp loop circuit** command.

You can choose one of three different kinds of help from the assistant, depending upon the **help** parameter value that you specify:

- **help full**—The assistant aids in both transmitting messages to and receiving messages from a remote node.
- **help transmit**—The assistant aids in transmitting loop messages to a remote node.
- **help receive**—The assistant aids in receiving loop messages from a remote node.

If you specify either the **assistant physical address** or **assistant node** parameter and do not specify the **help** parameter, you will receive **full** assistance by default. The three types of assistance are shown in Figures 5-4, 5-5, and 5-6.

Figure 5–4: Loopback Test Using Full Assistance

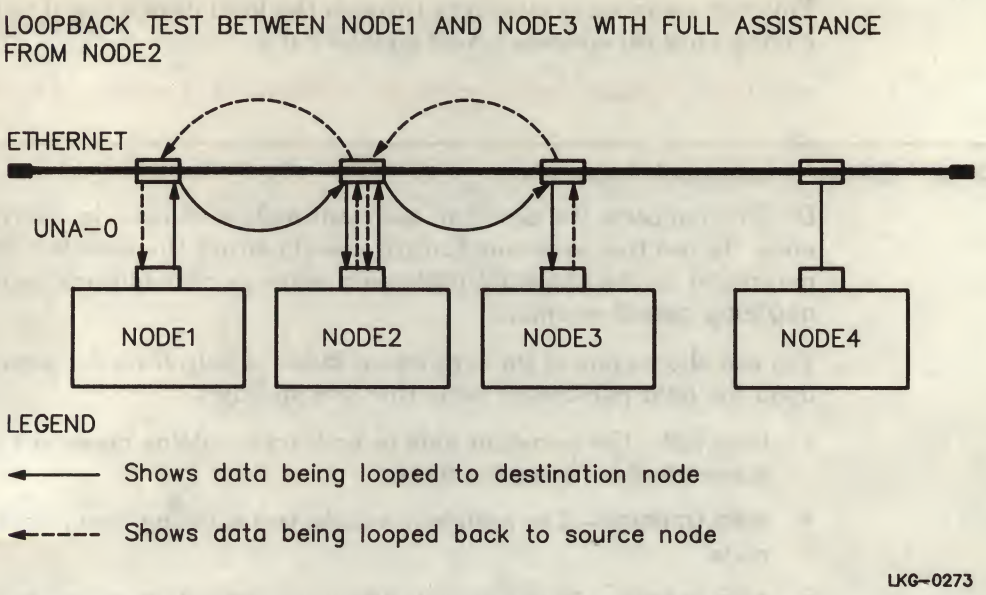


Figure 5–5: Loopback Test Using Transmit Assistance

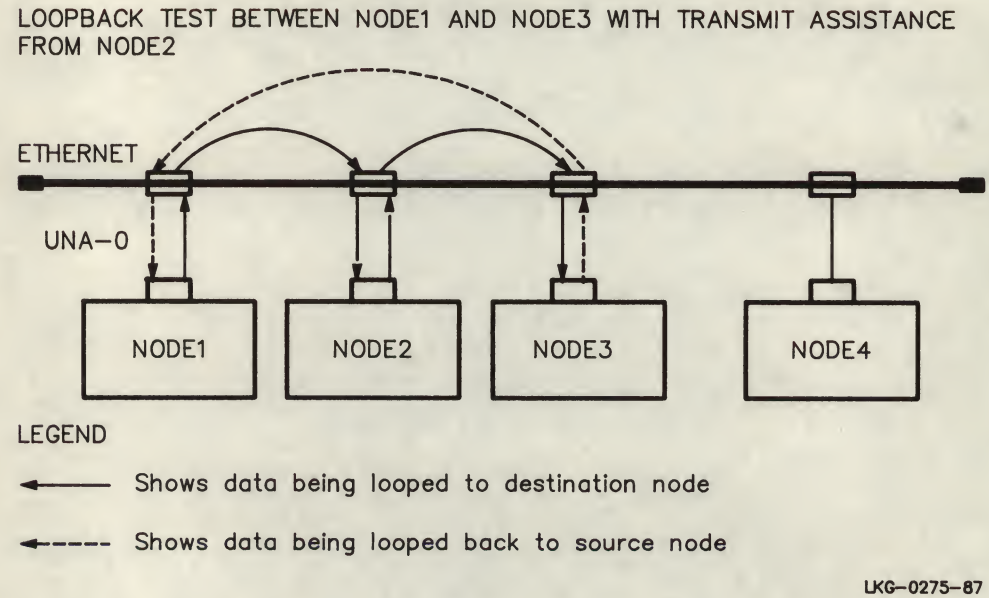
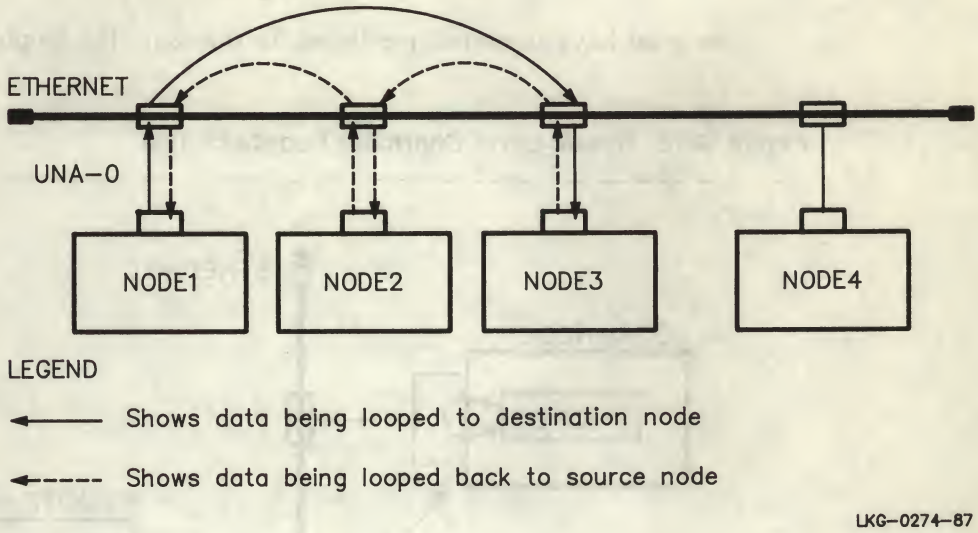


Figure 5–6: Loopback Test Using Receive Assistance



There are various reasons why you might choose one form of assistance over another. For example, if the target node to which you want to transmit a message is not receiving messages from your node, you can request assistance in transmitting messages to it. Similarly, if your node is able to transmit messages to the target node but not able to receive messages from it, you can send a message directly to the target node and request the assistant's aid in receiving a message back. If you encounter difficulties in both sending and receiving messages, you can request the assistant's aid in transmitting and receiving messages.

The following examples illustrate the use of the **assistant physical address** and **assistant node** parameters:

EXAMPLE 1:

In this command, you request the node described by Ethernet physical address AA-00-04-00-15-04 to assist you in testing the node described by Ethernet physical address AA-00-04-00-18-04. Since **assistant physical address** is specified without the **help** parameter, **full** assistance is given.

```
ncp> loop circuit una-0 physical address aa-00-04-00-18-04
 assistant physical address aa-00-04-00-15-04 RET
```

EXAMPLE 2:

In this command, you request node THRUSH to assist in testing node LOON by transmitting the loopback data to node LOON.

```
ncp> loop circuit una-0 node loon assistant node thrush help transmit RET
```

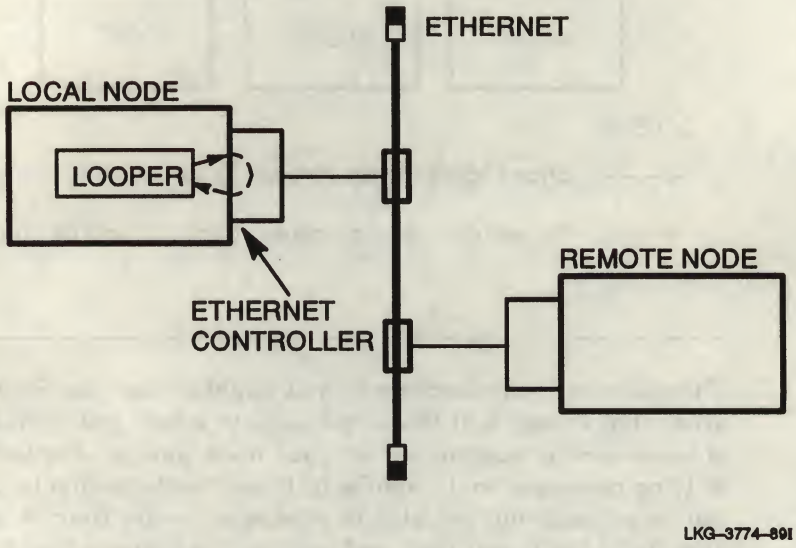
5.3.2 Controller Loopback Test

The controller loopback test verifies whether the circuit to the controller and the controller itself are functional. Figure 5-7 illustrates a controller loopback test.

NOTE

You must have superuser privileges for the controller loopback test.

Figure 5–7: Circuit-Level Controller Loopback Test



Before you begin this test, issue an **ncp show circuit summary** command to see whether the circuit state is **on**. If the circuit is off, use the **ncp set circuit state** command to turn it on.

To begin the controller loopback test, set the controller to **loopback** mode by using the **set line** command; then issue the **ncp loop circuit** command.

EXAMPLE:

This set of commands tests the circuit up to the controller for physical line una-0 connected to the local node by circuit una-0.

```
ncp>set line una-0 controller loopback RET
ncp>loop circuit una-0 count 10 length 32 RET
Loop succeeded
ncp>set line una-0 controller normal RET
```

If this test succeeds and the software loopback test fails, perform a circuit loopback test to see whether the device and device cable are functional.

5.4 The dts/dtr Tests

There are four basic tests provided by **dts/dtr**:

- Connect test
- Data test
- Disconnect test
- Interrupt test

Each test is divided into a set of subtests. The following sections describe the tests and subtests.

5.4.1 Connect Tests

Connect tests verify that the receiving node (**dtr** node) can process connection requests and return acceptance and rejection messages with or without optional user data. You can perform the following connect tests:

- **Connect reject without user data.** The **dts** node sends a connection request to the **dtr** node. The **dtr** node returns a rejection message that does not contain optional data.
- **Connect accept without user data.** The **dts** node sends a connection request to the **dtr** node. The **dtr** node returns an acceptance message that does not contain optional data.
- **Connect reject with 16 bytes of standard user data.** The **dts** node sends a connection request to the **dtr** node. The **dtr** node returns a rejection message that contains 16 bytes of optional data.
- **Connect accept with 16 bytes of standard user data.** The **dts** node sends a connection request to the **dtr** node. The **dtr** node returns an acceptance message that contains 16 bytes of optional data.
- **Connect reject with received user data used as reject user data.** The **dts** node sends a connection request that contains optional data to the **dtr** node. The **dtr** node returns a rejection message that contains the same optional data.
- **Connect accept with received user data used as accept user data.** The **dts** node sends a connection request that contains optional data to the **dtr** node. The **dtr** node returns an acceptance message that contains the same optional data.

5.4.2 Data Tests

Data tests provide a full range of tests from very simple data sink operations through data integrity checking. You can perform the following data tests:

- **Sink test.** The **dtr** program ignores all data received. No sequence or content validation is performed.
- **Sequence test.** Data messages transmitted by **dts** to **dtr** include a 4-byte sequence number. If a message is received out of sequence, **dtr** aborts the logical link and the test.

- **Pattern test.** Data messages transmitted to **dtr** have both a sequence number and a standard data pattern. If neither the sequence number nor the received data matches the expected data, **dtr** aborts the logical link and the test.
- **Echo test.** Data messages received by **dtr** are transmitted back to **dts**, checks the sequence number and data of the returned messages. If either is incorrect, **dts** aborts the link and the test.

5.4.3 Disconnect Tests

Disconnect tests verify that the receiving node (**dtr** node) can send out disconnect messages with or without optional user data. You can perform the following disconnect tests:

- **Disconnect without data.** The **dtr** node sends a disconnect message that does not contain user data to the **dts** node.
- **Abort without user data.** The **dtr** node sends an abort message that does not contain user data to the **dts** node.
- **Disconnect with 16 bytes of standard user data.** The **dtr** node sends a disconnect message that contains 16 bytes of optional data to the **dts** node.
- **Abort with 16 bytes of standard user data.** The **dtr** node sends an abort message that contains 16 bytes of optional data to the **dts** node.
- **Disconnect with received connect user data used as disconnect user data.** The **dts** node sends a message containing optional user data to the **dtr** node. The **dtr** node returns a disconnect message containing the same optional data to the **dts** node.
- **Abort with received connect user data used as abort user data.** The **dts** node sends a message containing optional user data to the **dtr** node. The **dtr** node returns an abort message containing the same optional data to the **dts** node.

5.4.4 Interrupt Tests

Interrupt tests provide a full range of test capabilities from very simple data sink operations through data integrity checking. Interrupt tests that the user can perform are as follows:

- **Sink test.** The **dtr** program ignores all interrupt data received. No sequence or content validation is performed.
- **Sequence test.** Interrupt messages transmitted by **dts** to **dtr** contain a 4-byte sequence number. If a message is received out of sequence, **dtr** aborts the logical link and the test.
- **Pattern test.** Interrupt messages transmitted to **dtr** have both a sequence number and a standard data pattern. If neither the sequence number nor the data pattern received matches the expected data, **dtr** aborts the logical link and the test.
- **Echo test.** Interrupt messages received by **dtr** are transmitted back to **dts**, which checks the sequence number and data of the returned messages. If either is incorrect, **dts** aborts the link and the test.

5.5 Performing dts/dtr Tests

The following procedure shows how to set up and run **dts/dtr** tests.

1. Be sure that the DECnet communications line is in the **on** state.
2. Enter the following command:

   % dts

   The system responds with a message like the following and a prompt:

   ```
   DTS initiated on Mon Feb 26 13:06:22 1990
   (DECnet-ULTRIX)

   DTS>
   ```

3. Enter **dts** commands at the command prompt.
4. To end testing, type **exit** in response to the **dts** prompt. The **dts** program prints a termination message on your screen when it exits, and the ULTRIX prompt reappears.

You can also enter **dts** commands with a **dts** command file.

EXAMPLE:

This command instructs **dts** to process the commands contained in the file **dtsshell** and to redirect the output to logging file **dts.log**.

```
% dts <dtsshell >dts.log
```

The exit status returned by **dts** commands is useful in a shell script.

5.5.1 Command Syntax for dts

Use the following format when entering **dts** commands:

dts>*test*[*qualifiers*][*test-specific-qualifiers*]

where

test specifies the type of test, which must be one of the following:

connect	connect test
disconnect	disconnect test
data	data test
interrupt	interrupt test

qualifiers specifies any number of the following optional qualifiers. Once specified, these qualifiers remain in effect for all applicable tests until you change them or exit from **dts**. Each qualifier must be preceded by a slash (/).

/nodename=*node-id* specifies the name or address of the DECnet node on which you want **dtr** to run. The default is the local node. If you run **dtr** on a remote node, you must run it on a default nonprivileged account (guest account), because you cannot specify access-control information with this qualifier.

/print or **/noprint** (default)	tells **dts** whether to print (log) test results.
/statistics or **/nostatistics** (default)	tells **dts** whether to print statistics on data and interrupt tests.
/display or **/nodisplay** (default)	tells **dts** whether to print the data and interrupt messages transmitted to **dtr**.
/speed=*number*	specifies the test line speed in bits per second (default=0); **dts** uses this data for reporting statistics.
test-specific-qualifiers	specifies any number of test-specific qualifiers, as defined in the following sections. Test-specific qualifiers apply to the current test only.

The command syntax described in this section uses the following conventions:

- All test names and qualifiers can be abbreviated to the first three or more unique characters.
- The default values for a qualifier remain in effect until a different value is specified. The specified value then becomes the new default for all following tests until that value is changed.

NOTE

Be sure to review the graphic conventions described in the Preface.

5.5.1.1 Connect Test Syntax

Use the following format to perform a **connect** test:

connect[*qualifiers*][*test-specific-qualifiers*]

where *test-specific-qualifiers* can be any of the following:

/type=*subtest*	specifies the type of test, where *subtest* can be:	
	accept	connect accept test (default)
	reject	connect reject test
/return=*type* or **noreturn**	specifies the type of data returned by **dtr**, where *type* can be:	
	standard	standard user data
	received	received user data
	noreturn	causes no optional user data to be returned

EXAMPLE:

This command invokes a **connect accept** test (by default) with remote node MONTRL.

```
dts>connect/nodename=montrl/return=received RET
```

The **dtr** program returns received user data as part of the test.

5.5.1.2 Disconnect Test Syntax

Use the following format to perform a **disconnect** test:

disconnect[*qualifiers*] [*test-specific-qualifiers*]

where *test-specific-qualifiers* can be any of the following:

type=subtest — specifies the type of test, where *subtest* can be:

synchronous	synchronous disconnect test
abort	disconnect abort test (default)

/return=*type*
/noreturn — specifies the type of data returned by **dtr**, where *type* can be:

standard	standard user data
received	received user data

The **/noreturn** qualifier causes no optional user data to be returned.

EXAMPLE:

This command invokes a **synchronous disconnect** test with remote node PARIS.

```
dts>disconnect/nodename=paris/type=synchronous RET
```

The **dtr** program will not return any optional user data.

5.5.1.3 Data Test Syntax

Use this format to perform a **data** test:

data[*qualifiers*] [*test-specific-qualifiers*]

where *test-specific-qualifiers* can be any of the following:

/type=*subtest* — specifies the type of test, where *subtest* can be:

sink	sink test (default)
sequence	sequence test
pattern	pattern test
echo	echo test

/size=*number* — specifies data message length in bytes, where *number* is a value from 1 to 2,048 (default=128). NOTE: The minimum value for *number* on a sequence test is 4 and on a pattern test is 5.

/test-duration — specifies duration of the test in one of the following formats:

seconds=*number*	range: 1 to 60
minutes=*number*	range: 1 to 60
hours=*number*	range: 1 to 24

The default is **seconds**=15.

/flow=*type* or
/noflow (default) — specifies type of flow control if any where *type* can be:

segment	segment flow control
message	message flow control (default, if **/flow** is specified)

If **dtr** is running on DECnet–ULTRIX software, it must use the system default.

/rqueueD=*number* — specifies number of pending receives for **dtr** to maintain, where *number* is a value from 1 to 16 (default = 1). If the remote system is a DECnet–ULTRIX node, this parameter is ignored.

/nak=*number* or /noack	specifies the number of segments between negative acknowledgments (NAKs). If the remote system is a DECnet–ULTRIX node, this parameter is ignored.
/back=*number* or */noback*	specifies the number of segments before back pressuring. If the remote system is a DECnet–ULTRIX node, this parameter is ignored.

EXAMPLE:

This command invokes the **data** test with the sink subtest (by default). The **dts** program sends messages to **dtr** on node JONES (by default from a previous command). The message size is 512 bytes, and the duration of the test is 30 seconds.

```
dts> data/size=512/seconds=30 RET
 DTS --I-- Test started at 11:23:30
 DTS --I-- Test finished at 11:24:00

Test parameters:
     Target node           "jones"
     Test duration (sec)   30
     Message size (bytes)  512)

Summary statistics:
     Total messages SENT   48
     Total bytes SENT      24576
     Messages per second   1.60
     Bytes per second      819.20
     Line throughput (baud)  6553
```

5.5.1.4 Interrupt Test Syntax

Use this format to perform an **Interrupt** test:

Interrupt[*qualifiers*] [*test-specific-qualifiers*]

where *test-specific-qualifiers* can be any of the following:

/type=*subtest*	specifies the type of test, where subtest can be: **sink** sink test (default) **sequence** sequence test **pattern** pattern test **echo** echo test
/size=*number*	specifies data message length in bytes, where *number* is a value from 1 to 16 (default = 16). NOTE: The minimum value for *number* on a sequence test is 4 and on a pattern test is 5.
test-duration	specifies duration of the test in one of the following formats: **seconds**=*number* range: 1 to 60 **minutes**=*number* range: 1 to 60 **hours**=*number* range: 1 to 24 The default is **seconds**=15.
/rqueue=*number*	specifies number of pending receives for **dtr** to maintain, where *number* is a value from 1 to 16 (default=1). If the remote system is DECnet–ULTRIX, this parameter is ignored.

EXAMPLE:

This command invokes the **interrupt** test with the pattern subtest. The **dts** program sends interrupt messages to **dtr** on node DALLAS, where test information is to be printed. The default is used for message size, and the duration of the test is 30 seconds.

```
dts> interrupt/nodename=dallas/print/type=pat/seconds=30 RET

 DTS --I-- Test started at 17:44:10
 DTS --I-- Test finished at 17:44:40

Test parameters:

Target node                "dallas"
Test duration (sec)        30
Message size (bytes)       16

Summary statistics:
Total messages SENT        2734
Total bytes SENT           43744
Messages per second        91.1
Bytes per second           1458
Line throughput (baud)     11665
```

Chapter 6

DECnet–ULTRIX Network Maintenance Commands

This chapter describes two DECnet–ULTRIX maintenance commands. The format for this information corresponds to that in the ULTRIX reference pages. See the *ULTRIX Reference* manuals for more information on format.

The name of each command being described appears in a running head, followed by the appropriate section number and suffix in parentheses. For example, **dts(8dn)** appears on the reference pages that describe **dts**. The number **8** indicates that the section describes maintenance commands. The **dn** suffix indicates that the commands are used in the DECnet domain.

The command descriptions use the graphic conventions described in the Preface.

Table 6-1 summarizes the functions of the DECnet–ULTRIX maintenance commands.

Table 6–1: DECnet–ULTRIX Maintenance Commands

Command	Function
dts	Evokes the DECnet Test Sender.
ncp	Runs the Network Control Program.

The following command descriptions also appear on-line in the **dts(8dn)** and **ncp(8dn)** manual pages.

dts (8dn)

NAME

dts — evoke the DECnet test sender

SYNTAX

dts [**test**[*/qualifiers*][*/test-specific-qualifiers*]]

where

test is the name of the test you want to run:

connect	specifies a connect test. These tests verify that the **dtr** node can process connection requests and return **accept** and **reject** messages with or without optional data.
data	specifies a data test. These tests include sink, sequence, pattern, and echo tests.
disconnect	specifies a disconnection test. These tests verify that the node can send out disconnection messages with or without optional data.
interrupt	specifies an interrupt test. These tests include sink, sequence, pattern, and echo tests for interrupt data.

qualifiers are the valid qualifiers for this command. Once specified, these qualifiers remain in effect for all applicable tests until you either change them or exit from **dts**. Precede each qualifier with a slash (/). You can specify any number of these qualifiers:

nodename=*node-id*	specifies the name or address of the DECnet node on which you want **dtr** to run. The default is the local node. If you run **dtr** on a remote node, you must run it on a default nonprivileged account (guest account) because you cannot specify access-control information with this qualifier.
print or **noprint** (default)	tells **dts** whether or not to print test results.
statistics or **nostatistics** (default)	tells **dts** whether or not to print statistics on data and interrupt tests.
display or **nodisplay** (default)	tells **dts** whether or not to print the data and interrupt message transmitted to **dtr**.
speed=*number*	specifies the test line speed in bits per second (default = 0). The **dts** program uses this data for reporting statistics.

test-specific-qualifiers are the valid qualifiers for the test you specify. Test-specific qualifiers apply only to the current test.

DESCRIPTION

The **dts** utility is the DECnet–ULTRIX transmitter test program that runs four tests: connect, data, disconnect, and interrupt.

Specify the test you want in one of two ways:

% dts *test[/qualifiers][/test-specific-qualifiers]

or

% dts
dts> test[/qualifiers][/test-specific-qualifiers]

The **dts** program initiates each test by issuing a connect request to the **dtr** program. Parameter information associated with each type of test requested is passed by **dts** to **dtr** during a connection.

EXAMPLE

The following command invokes a data test with remote node OHLONE. Data messages of 512 bytes are transmitted to the **dts** program on the remote node, which then echos them back. The test runs for 10 seconds. Test parameters and summary statistics are displayed after the test completes.

```
% dts data/nodename=ohlone/type=echo/size=512/seconds=10 RET
```

ncp (8dn)

NAME

ncp — run the Network Control Program

SYNTAX

ncp [*command*] [*component*] [*parameter list*]

where

command specifies the command you want to execute:

help [*command...*]	displays a short description of the command specified in the argument list. If no arguments are given, it displays a list of the recognized commands.
show	displays information about the volatile database.
list	displays information about the permanent database.
set	creates or modifies parameters in the volatile database. Also specifies a new node as the executor.
define	creates or modifies parameters in the permanent database.
clear	removes parameters from the volatile database.
purge	removes parameters from the permanent database.
zero	resets the DECnet counters, including line and circuit counters.
tell	sends **ncp** commands to a remote node for execution.
loop	tests either a node in the network or a circuit on the executor node.

For a complete list of **ncp** commands and their use in a DECnet–ULTRIX environment, see the *DECnet–ULTRIX NCP Command Reference* manual.

DESCRIPTION

The DECnet–ULTRIX Network Control Program (**ncp**) is a network management utility. You can use **ncp** commands to configure, control, monitor, and test DECnet nodes.

To execute **ncp**, use one of the following formats:

% **ncp** *command*

or

% **ncp**
ncp> *command*

RESTRICTION

You must have superuser privileges to use an **ncp** command that modifies a database. However, any user can use the **ncp help**, **show**, and **list** commands to display information from the permanent and volatile databases.

EXAMPLE

The following example starts the DECnet–ULTRIX software running on your local system:

```
% ncp set executor state on RET
```

The following example sends the **show executor characteristics** command to the remote DECrouter 200 node named ROUTER, where the command is then executed. The information about ROUTER is displayed at the local node.

```
% ncp tell router show executor characteristics RET
```

Index

A

C

D

E

F

Q

R

S

T

U

V

W

HOW TO ORDER ADDITIONAL DOCUMENTATION

DIRECT TELEPHONE ORDERS

In Continental USA
call 800–DIGITAL

In Canada
call 800–267–6215

In New Hampshire
Alaska or Hawaii
call 603–884–6660

In Puerto Rico
call 809–754–7575

ELECTRONIC ORDERS (U.S. ONLY)

Dial 800–DEC–DEMO with any VT100 or VT200
compatible terminal and a 1200 baud modem.
If you need assistance, call 1–800–DIGITAL.

DIRECT MAIL ORDERS (U.S. and Puerto Rico*)

DIGITAL EQUIPMENT CORPORATION
P.O. Box CS2008
Nashua, New Hampshire 03061

DIRECT MAIL ORDERS (Canada)

DIGITAL EQUIPMENT OF CANADA LTD.
940 Belfast Road
Ottawa, Ontario, Canada K1G 4C2
Attn: A&SG Business Manager

INTERNATIONAL

DIGITAL
EQUIPMENT CORPORATION
A&SG Business Manager
c/o Digital's local subsidiary
or approved distributor

Internal orders should be placed through the Software Distribution Center (SDC),
Digital Equipment Corporation, Westminster, Massachusetts 01473

*Any prepaid order from Puerto Rico must be placed
with the Local Digital Subsidiary:
809–754–7575

DECnet–ULTRIX
Network Management
AA–EE38D–TE

READER'S COMMENTS

What do you think of this manual? Your comments and suggestions will help us to improve the quality and usefulness of our publications.

Please rate this manual:

	Poor				Excellent
Accuracy	1	2	3	4	5
Readability	1	2	3	4	5
Examples	1	2	3	4	5
Organization	1	2	3	4	5
Completeness	1	2	3	4	5

Did you find errors in this manual? If so, please specify the error(s) and page number(s).

_____

_____

_____

_____

General comments:

_____

_____

_____

_____

Suggestions for improvement:

_____

_____

_____

_____

Name _____ Date _____

Title _____ Department _____

Company _____ Street _____

City _____ State/Country _____ Zip _____

DO NOT CUT – FOLD HERE AND TAPE

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY LABEL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

digital™

Networks and
Communications Publications
550 King Street
Littleton, MA 01460–1289

DO NOT CUT – FOLD HERE

CUT ON THIS LINE

digital

DECnet–ULTRIX

NCP Command Reference

May 1990

This manual describes the Network Control Program (ncp) commands you use to define, monitor, and test your network.

Supersession/Update Information:	This is a new manual.
Operating System and Version:	ULTRIX V4.0
Software Version:	DECnet–ULTRIX V4.0

digital™

Order Number: AA–PB62A–TE

AA–PB62A–TE
May 1990

This manual was produced by Networks and Communications Publications.

Contents

Figures

Tables

Preface

This manual explains how to use the Network Control Program (ncp) commands to manage a Phase IV DECnet–ULTRIX node within the DECnet environment.

Manual Objectives

This manual describes the ncp commands that you can use to configure, monitor, and test your network. Additional reference information summarizes the DECnet–ULTRIX network objects, event logging, network counters, and Ethernet addressing information.

Intended Audience

This manual is for anyone responsible for configuring, maintaining, and managing the network. The manual refers to all such people as the network manager.

Structure of This Manual

This manual is divided as follows:

Chapter 1	Gives a detailed explanation of how to invoke the Network Control Program and gives guidelines for using ncp commands.
Chapter 2	Describes each ncp command, including its function, syntax, parameters, and gives an example.
Chapter 3	Lists and explains all error messages that can occur from the execution of an ncp command.
Chapter 4	Lists and explains the network management event-logging messages.
Chapter 5	Lists and explains all network counters maintained by the DECnet–ULTRIX software.
Appendix A	Summarizes all ncp commands and their parameters.
Appendix B	Defines all DECnet–ULTRIX objects by name, number, file, and function.
Appendix C	Describes physical and multicast addressing for nodes on Ethernet lines.

Related Documents

For more information about DECnet-ULTRIX software, see the following manuals:

- *DECnet–ULTRIX Release Notes*

  Contains information and updates not included in the DECnet–ULTRIX documentation set.

- *DECnet–ULTRIX Installation*

  Contains step-by-step procedures for installing your DECnet–ULTRIX software and testing your node's operation in the network.

- *DECnet–ULTRIX Use*

  Describes the DECnet–ULTRIX user commands and explains how to use them to perform file transfer and other user tasks.

- *DECnet–ULTRIX Network Management*

  Introduces the network manager to DECnet databases and components and describes how to use the Network Control Program (**ncp**) to configure, monitor, and test these components.

- *DECnet–ULTRIX Programming*

  Describes the DECnet–ULTRIX system calls and subroutines. Also provides information about application programming within the DECnet environment and contains supplemental information for programming the DECnet–ULTRIX socket interface.

- *DECnet–ULTRIX DECnet-Internet Gateway Use and Management*

  Describes the DECnet-Internet Gateway and explains how to use, manage, and install it.

To obtain a detailed description of the Digital Network Architecture, refer to the following document:

- *DECnet Digital Network Architecture (Phase IV), General Description*

Acronyms

The following acronyms are used in this manual:

DNA	Digital Network Architecture
dtr	DECnet Test Receiver
dts	DECnet Test Sender
ECL	End Communication layer
evl	Event Logger
fal	File Access Listener
mir	Loopback Mirror
ncp	Network Control Program
nml	Network Management Listener
NSP	Network Services Protocol

Conventions Used in This Manual

Convention	Meaning
`Example`	Examples appear in this special type.
Red type	Red type in examples indicates text that a user enters.
special	All commands and parameters appear in **special type**.
lowercase	If a command appears in lowercase type in a command format or in an example, you must enter it in lowercase.
italic	Italic type in command formats and system displays indicates a variable, for which either you or the system must supply a value.
{ }	Braces indicate that you must specify one of the enclosed options, but no more than one. Do not type the braces when you enter the command.
[ ]	Square brackets indicate that you can use one, and only one, of the enclosed options. Do not type the brackets when you enter the command.
( )	Parentheses enclose a set of options that you must specify together or not at all.
Vertical list of options	A vertical list of options not enclosed within braces, brackets, or parentheses indicates that you can specify any number of options, or none at all.
key	This is a symbol for a keyboard key. CTRL/*key* represents a CTRL key sequence, where you press the CTRL key at the same time as the specified key.
%	This is the default user prompt in multiuser mode.
#	This is the default superuser prompt.

All Ethernet addresses are hexadecimal; all other numbers are decimal unless otherwise noted.

Chapter 1

Understanding the Network Control Program

This chapter tells you how to use the Network Control Program (**ncp**) on DECnet–ULTRIX nodes in the following ways:

- Invoke and exit the Network Control Program.
- Execute **ncp** commands.
- Issue **ncp** commands from your terminal for execution at a remote node.
- Maintain network security with superuser privileges.
- Down-line load a remote node using **load** and **trigger** commands.

1.1 Getting Started with ncp

The Network Control Program lets you issue **ncp** commands from a terminal or from a shell script. You can execute most **ncp** commands either locally or remotely.

1.1.1 Invoking ncp

You can invoke **ncp** in three ways:

- Enter **ncp** at the prompt.

  ```
  % ncp RET
  ```

  The program then prompts you as follows:

  ```
  ncp>
  ```

  Enter your **ncp** command after the prompt and press RET.

- Enter an entire **ncp** command line, for example:

  ```
  % ncp show known circuits counters RET
  ```

  where **show known circuits counters** is a valid **ncp** command. After the command executes, you return to the shell.

- Enter **ncp** with a shell script, for example:

  ```
  % ncp <scripta
  ```

  where *scripta* is the name of a shell script that contains a sequence of **ncp** commands. Your shell script can use the exit status returned by **ncp** commands.

The following example shows a sample shell script:

```
ncp sho line una-0
if ( $status != 0 ) then
        echo ""
        echo "This ncp command failed."
        echo ""
endif
```

This sample shell script uses the exit status from an **ncp** command to determine whether or not to echo a message. If the **ncp show line** command fails, the shell script echoes the message.

NOTE

You can insert comment lines in an **ncp** shell script by prefacing each comment line with a pound sign (#).

1.1.2 Exiting ncp

To exit **ncp**, use either the **exit** command, the **quit** command, or CTRL/D at the **ncp** prompt.

1.2 Executing ncp Commands

The following sections explain the **ncp** command syntax and procedures for executing them.

1.2.1 Command Syntax

Most commands consist of three parts: the command verb, a component on which the command operates, and one or more parameters that further qualify the action to be taken on the component. You enter a command at the **ncp** prompt in the following order:

ncp *command-verb component parameter*

EXAMPLE:

This example shows a **list** command.

```
list line una-0 characteristics RET
```

For each command, you must supply one verb and one component option. The number of parameters that you can supply varies with each command.

Some commands have a list of optional parameters, any number of which you can specify. For example, the **list line** command lets you select one or more of the following parameters:

list { **line** *line-id* | **known lines** } [**characteristics** | **counter** | **status** | **summary**]

Some commands have the **all** parameter. When you specify **all**, you cannot specify any other parameters for that command. If you do not specify **all**, you can use any number of the remaining parameters. For example, you can use **all** or you can specify any other **clear node** command parameters:

clear { **node** *node-id* / **known nodes** } [**all** / **diagnostic file** / **dump file** / **hardware address** / **host** / **load file** / **name** / **secondary loader** / **service circuit** / **service password** / **tertiary loader**]

EXAMPLE:

This example shows how you can use **all** to specify all **clear node** parameters for an object.

```
ncp>clear node NAVAHO all RET
```

Some commands have a bracketed list of optional parameters of which you can specify only one option. For example, the **show object** command lets you select only one parameter for a specified object or for all known objects.

show { **objects** *object-name* / **known objects** } [**characteristics** / **summary**]

1.2.2 Typing Command Lines

Enter the command keywords separated by spaces. You can abbreviate any keyword to the shortest number of unique characters that **ncp** accepts. For example, the following versions of the same command are equally valid:

```
ncp>show logging console RET
ncp>sho log con RET
ncp>sh lo c RET
```

1.2.3 Using the Help Facility

Enter **help** at the **ncp** prompt for assistance in selecting network management commands and parameter options. The **ncp** utility returns a list of subjects for which help information is available.

EXAMPLE:

This example shows how the **help** command displays available information.

```
ncp>help RET

Help available for:
clear           command         define          exit
help            list            load            loop
parameters      prompting       purge           set
show            tell            trigger         zero

Topic?
```

For additional information, enter one of the command verbs at the **Topic?** prompt.

1.2.4 Command Prompting

Command prompting provides on-line assistance when you are entering **ncp** commands. If you press RET at the **ncp** prompt, **ncp** displays all legal options for that command keyword.

EXAMPLE:

This example shows you the kind of information you must provide when you press RET after entering the **show** command.

```
>ncp RET
ncp>show RET

(active, adjacent, area, circuit, executor, known, line,
logging, loop, module, node, object):
```

If you enter **known** at the colon (:) prompt, **ncp** prompts you for additional information, as shown in the following example:

```
(active, adjacent, area, circuit, executor, known, line,
logging, loop, module, node, object): known

(areas, circuits, lines, logging, nodes, objects):
```

If you enter an incomplete **set** or **define** command, **ncp** prompts you individually for each possible parameter. You can respond in one of several ways:

- Enter RET to omit the current parameter and to be prompted for the next one.
- Enter a parameter value and press RET to set this parameter and to be prompted for the next one.
- Enter a period (.) and press RET when you have specified the parameters that you want and are ready to exit the prompting loop to execute the command.
- Enter CTRL/D to cancel the entire command.

1.2.5 Error Reporting

If a command executes successfully, the **ncp** prompt appears on the next line. If a command does not complete successfully, an error message is displayed to indicate the reason for the failure. Chapter 3 lists **ncp** error messages.

1.2.6 Event Logging

The logging monitor interface from the DECnet event-logging facility provides a mechanism by which a user-written program can process network events. You must specify the name of the monitoring program and the events to be logged by using the following **ncp set logging** commands:

set logging monitor name *name* **state on**

set logging monitor events *event-list*

where

name	Is the file descriptor of the program to receive the event information (default: **evl**).

event-list Identifies one or more event classes and types to be logged. (See Chapter 4 for a list of event classes and types.)

NOTE

The event monitor facility can be used in conjunction with event logging, either on the console or to a file.

When the monitor program is **evl** (default), events are logged at the logging console. If you write your own monitor program to process network events, your program starts when the network starts. When the network starts up, **evl** passes two arguments to your monitor program:

Argument 1 — your monitor program file name

Argument 2 — the file descriptor of a pipe from which to read

Your monitor program reads the events in ASCII from the pipe and then processes them according to your specifications.

1.3 Executing ncp Commands Remotely

You can use **ncp** to modify parameters or display information at a remote node. You can execute **ncp** commands on a remote node while logged in to your local node by using either the **ncp tell** prefix or the **ncp set executor node** command.

Before attempting to execute an **ncp** command remotely:

- Refer to network management documentation for the remote node to see which commands it supports.
- Check for any access restrictions.

1.3.1 Using the tell Prefix

To execute a single **ncp** command at a remote node, issue the command with the **tell** prefix. The following example shows how you can use the **tell** prefix to send the **ncp** command **show executor node** to a remote node NAVAHO.

```
ncp>tell navaho show exec node char RET
```

1.3.2 Using the set executor Command

To execute a series of **ncp** commands at a remote node, use the **set executor node** command to set the specified remote node as the executor. Subsequent commands that you issue are executed at that node until you restore control to your own node by issuing a **clear executor node** command. If you exit **ncp** while the executor is set to a remote node, control is automatically returned to the local node when you reenter **ncp**. For example:

```
% ncp RET
ncp>set executor 2.95 RET
show executor characteristics
  .
  .
  .
```

```
define executor address 2.95
set executor incoming timer
  .
  .
  .
ncp>clear executor node RET
```

In this example, 2.95 is the node address.

1.3.3 Access Restrictions on Remote Nodes

Before you issue commands to be executed at a remote node, you may have to supply access control information. Depending on the types of access restrictions set up by the system manager of the remote node, you can gain access to a remote node from a DECnet–ULTRIX node in the following ways:

- Append access control information to the node ID in the **ncp** command string.
- Include access control information in an "alias" definition for the node.
- Use proxy verification on the remote node.

For the procedure on how to supply access control information, see the *DECnet–ULTRIX Use* manual.

1.4 Network Management Privileges

You must have superuser privileges to use an **ncp** command that modifies a database. However, anyone can exercise the **ncp show** or **list** commands locally to display component information from the databases.

On ULTRIX systems, some **ncp** commands (such as **purge**, **define**, **set**, and **zero**) require superuser privileges. Other systems may also require privileges for the same **ncp** commands. For example, some DECnet–VAX NCP commands require system privileges (SYSPRV), others require operator privileges (OPER), and still others do not require privileges. To determine the privileges you need to issue commands at either a DECnet–ULTRIX node for remote execution or a remote node, see the Network Control Program documentation for the remote node.

1.5 Issuing load and trigger Commands

When you issue the **load** command, the load host must have service enabled on its Ethernet circuit or it cannot perform a down-line load. When you issue the **trigger** command, potential load hosts must have service enabled on their Ethernet circuits or they cannot perform a down-line load.

To enable service, use the following command format:

set circuit *circuit-id* **service enabled**

where *circuit-id* identifies the Ethernet circuit for the host.

On the command line, enter either the DECnet node name or the DECnet node address of the server. The **load** and **trigger** commands have a similar syntax:

load node *node-name*
trigger node *node-name*

or

load node *node-address*
trigger node *node-address*

The following examples use the **load** command to load a node named NAVAHO with a node address of 55.1024.

```
ncp>load node NAVAHO  RET
ncp>load node 55.1024  RET
```

If the remote node you want to load has an enabled password, you must specify this password on the **load** and **trigger** command lines. For example, to load node NAVAHO with service password FF55, type:

```
ncp>load node NAVAHO service password FF55  RET
```

or

```
ncp>load node 55.1024 service password FF55  RET
```

or

```
ncp>trigger node NAVAHO service password FF55  RET
```

or

```
ncp>trigger node 55.1024 service password FF55  RET
```

Chapter 2

NCP Command Descriptions

This chapter gives detailed descriptions and examples of each **ncp** command. Table 2-1 summarizes **ncp** command functions:

Table 2–1: ncp Command Functions

Command	Function
clear circuit	Removes specified circuit parameters from the volatile database.
clear executor	Resets specified executor parameters to their default values in the volatile database.
clear executor node	Resets the default executor to the local node.
clear logging	Removes specified logging parameters from the volatile database.
clear node	Removes names associated with the nodes or removes specified parameters from the volatile database.
clear object	Removes specified objects from the volatile database.
define	Creates or modifies specified parameters in the permanent database.
define circuit	Modifies specified circuit parameters in the permanent database.
define executor	Modifies specified executor node parameters in the permanent database.
define line	Modifies specified line parameters in the permanent database.
define logging	Creates or modifies logging component parameters in the permanent database.
define node	Changes the name or address associated with a node or resets specified node parameters in the permanent database.
define object	Creates or modifies parameters for the specified objects in the permanent database.
help	Displays information about **ncp** commands and parameters.
list	Displays specified parameters in the permanent database.
list circuit	Displays specified circuit information stored in the permanent database.
list executor	Displays specified local node information stored in the permanent database.

(continued on next page)

Table 2–1 (Cont.): ncp Command Functions

Command	Function
list line	Displays specified line information stored in the permanent database.
list logging	Displays specified logging information stored in the permanent database.
list node	Displays specified node information stored in the permanent database.
list object	Displays specified object information stored in the permanent database.
load node	Down-line loads a specified remote node on the same Ethernet.
load via	Down-line loads a remote node on the same Ethernet by way of the specified circuit.
loop circuit	Tests a circuit between the executor and another node in the network.
loop node/executor	Tests a node in the network.
purge	Removes specified parameters in the permanent database.
purge circuit	Removes specified circuit parameters in the permanent database.
purge executor	Resets specified executor parameters to their default values in the permanent database.
purge logging	Removes specified logging parameters in the permanent database.
purge node	Removes the names associated with the nodes or removes specified node parameters from the permanent database.
purge object	Removes specified objects from the permanent database.
set	Creates or modifies specified parameters in the volatile database.
set circuit	Modifies specified circuit parameters in the volatile database.
set executor	Creates or modifies specified executor node parameters in the volatile database.
set executor node	Sets a default executor for subsequent ncp commands.
set line	Modifies specified line parameters in the volatile database.
set logging	Modifies logging component parameters in the volatile database.
set node	Changes the name or address associated with a node or resets specified node parameters in the volatile database.
set object	Creates or modifies parameters for specified objects in the volatile database.
show	Displays specified parameters in the volatile database.
show circuit	Displays specified circuit information stored in the volatile database.
show executor	Displays specified local node information stored in the volatile database.
show line	Displays specified line information stored in the volatile database.

(continued on next page)

Table 2–1 (Cont.): ncp Command Functions

Command	Function
show logging	Displays specified logging information stored in the volatile database.
show node	Displays specified node information stored in the volatile database.
show object	Displays specified object information stored in the volatile database.
tell	Sends an ncp command to a remote node for execution.
trigger	Starts the bootstrap operation of a remote node on the same Ethernet so that the node multicasts a request for a down-line load.
trigger via	Starts the bootstrap operation of a remote node on the same Ethernet so that the node multicasts a request for a down-line load through a specified circuit.
zero	Sets counters to zero.
zero circuit	Sets circuit counters to zero for the specified circuit.
zero executor	Sets node counters associated with and maintained on the executor node to zero.
zero line	Sets line counters to zero for the specified line.
zero node	Sets line counters to zero for the specified node. The executor node maintains node counters on a per-node basis.

clear circuit

DESCRIPTION

Removes specified circuit parameters from the volatile database.

RESTRICTION

You must have superuser privileges to execute this command.

SYNTAX

clear circuit *circuit-id* [**receive password** | **transmit password**]

where

circuit *circuit-id*	Specifies the circuit for which parameters are to be removed.
receive password	Removes the circuit's receive password.
transmit password	Removes the circuit's transmit password.

EXAMPLE

This command deletes all parameters for circuit una-0 from the volatile database.

```
ncp>clear circuit una-0 RET
```

clear executor

DESCRIPTION

Resets specified executor parameters to their default values in the volatile database.

RESTRICTION

You must have superuser privileges to execute this command.

SYNTAX

clear executor { **identification** | **incoming timer** | **outgoing timer** }

where

identification	Removes the text identification string for the executor node.
incoming timer	Resets the incoming timer to its default value.
outgoing timer	Resets the outgoing timer to its default value.

EXAMPLE

This command resets the local node's incoming timer to its default value in the volatile database.

```
ncp>clear executor incoming timer RET
```

clear executor node

DESCRIPTION

Resets the default executor to the local node.

RESTRICTION

Do not use the **tell** prefix with this command.

SYNTAX

clear executor node

EXAMPLE

This command returns **ncp** command execution from a remote node to the local node.

```
ncp>clear executor node RET
```

clear logging

DESCRIPTION

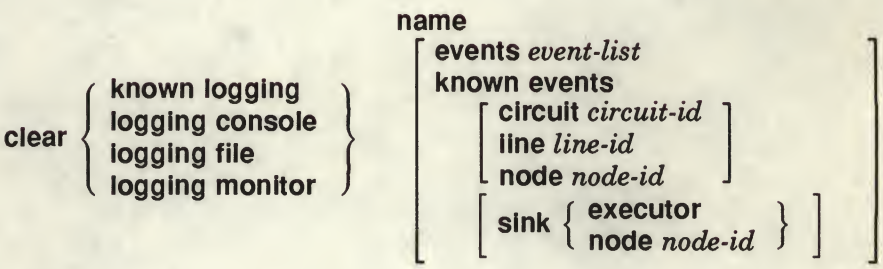
Removes specified logging parameters from the volatile database.

RESTRICTIONS

You must have superuser privileges to execute this command.

Whenever you specify a *circuit*, *line*, *node*, or *sink* in a **clear logging** command, you must also include an **events** list or **known events** parameter.

SYNTAX



where

circuit *circuit-id*
Inhibits event logging for the specified circuit.

events *event-list*
Removes the event class and types specified in *event-list* for the specified component.

known events
Removes all known events that DECnet–ULTRIX can generate for the specified component.

known logging
Removes parameters for all known logging components.

line *line-id*
Inhibits event logging for the specified line.

logging console
Removes parameters for the console-logging component.

logging file
Removes parameters for the file-logging component.

logging monitor
Removes parameters for the monitor-logging component.

name
Returns the specified logging component to its default name (console: **/device/console**; file: **/usr/adm/eventlog**; monitor **evl**.)

node *node-id*
Inhibits event logging for the specified node.

sink
Inhibits logging of the specified events at the specified node. You must specify one of the following qualifiers:

executor
This is the default setting. Events are not to be logged at the executor node.

node *node-id*
Events are not to be logged at the specified node.

EXAMPLE

This command ceases logging of event 2.1 to the console.

```
ncp>clear logging console event 2.1 RET
```

clear node

DESCRIPTION

Removes names associated with the nodes or removes specified node parameters from the volatile database.

RESTRICTION

You must have superuser privileges to execute this command.

SYNTAX

clear { **node** *node-id* | **known nodes** } { **diagnostic file** | **dump file** | **hardware address** | **host** | **load file** | **name** | **secondary loader** | **service circuit** | **service password** | **tertiary loader** }

or

clear { **node** *node-id* | **known nodes** } **all**

where

all	Removes all parameters for the specified node(s) so that the network no longer recognizes the nodes. Use of **all** precludes the use of any other parameters.
diagnostic file	Removes the identification of the down-line load diagnostic file.
dump file	Removes the up-line dump file identification.
hardware address	Removes the Ethernet address of the system hardware.
host	Removes the host node identification.
known nodes	Performs the specified function for all known nodes.
load file	Removes the identification of the down-line load file.
name	Removes the node name(s) associated with the specified node(s).
node *node-id*	Performs the specified function for the specified node only.
secondary loader	Removes the identification of the secondary down-line loading file.
service circuit	Removes the circuit parameter associated with the node for down-line loading purposes.
service password	Removes the password parameter. This password parameter is required to trigger the bootstrap mechanism of the node to be down-line loaded.

tertiary loader	Removes the identification of the tertiary down-line loading file.

EXAMPLE

This command removes all information for node BOSTON from the volatile database.

```
ncp>clear node boston all RET
```

clear object

DESCRIPTION

Removes specified objects from the volatile database.

RESTRICTION

You must have superuser privileges to execute this command.

SYNTAX

clear { **object** *object-name* | **known objects** }

where

object *object-name*	Specifies the object for which parameters are to be removed.
known objects	Specifies that parameters are to be removed for all known objects.

EXAMPLE

This command removes the network terminal handler (**dtermd**) from the volatile database.

```
ncp>clear object dtermd RET
```

define circuit

DESCRIPTION

Modifies specified circuit parameter(s) in the permanent database.

RESTRICTION

You must have superuser privileges to execute this command.

SYNTAX

define circuit *circuit-id* { **hello timer** *seconds* | **receive password** *password* | **state** *circuit-state* { **off** | **on** } | **transmit password** *password* }

where

circuit *circuit-id*
Specifies the circuit for which parameters are to be modified.

hello timer *seconds*
Specifies the frequency of routing hello messages sent to adjacent nodes on the circuit. The range is 1 through 8191.

receive password *password*
Specifies a 1- to 8-character ASCII password associated with the circuit that the executor expects to receive from the remote node during a routing initialization sequence.

state *circuit-state*
Specifies the circuit's operational state. There are two possible states:

- **off** — The circuit is not in use.
- **on** — The circuit is available for normal use or service functions.

transmit password *password*
Specifies a 1- to 8-character ASCII password associated with the circuit that the executor sends to the remote node during a routing initialization sequence.

EXAMPLES

This command makes circuit una-0 unavailable for use.

```
ncp>define circuit una-0 state off RET
```

define executor

DESCRIPTION

Creates or modifies specified executor node parameters in the permanent database.

NOTE

If you use the **define executor** command to issue a series of commands at a remote node, you can use the **tell** prefix to issue an **ncp** command to yet another remote node.

RESTRICTION

You must have superuser privileges to execute this command.

SYNTAX

define executor { **address** *node-address* | **delay factor** *number* | **delay weight** *number* | **gateway access** { **disable** | **enable** } | **gateway user** *login-name* | **identification** *id-string* | **inactivity timer** *seconds* | **incoming proxy** { **disable** | **enable** } | **incoming timer** *seconds* | **maximum links** *number* | **maximum node counters** *number* | **name** *node-name* | **outgoing proxy** { **disable** | **enable** } | **outgoing timer** *seconds* | **pipeline quota** *number* | **retransmit factor** *number* | **segment buffer size** *number* }

where

address *node-address*
: Specifies the address of the executor node. (Remember, **define** commands do not take effect until the next time you start up the network.)

delay factor *number*
: Specifies a number to multiply times 1/16 of the estimated round-trip delay to a node in order to set the retransmission timer for that node. The range is 1 through 255.

delay weight *number*
: Specifies a value to apply to a current round-trip delay to a remote node in order to update the estimated round-trip delay time. The range is 1 through 255.

gateway access
Specifies whether or not the executor allows the DECnet objects **fal** and **dlogind** to provide access to the Internet network. Choose one of the following modes:

- **disable** — Turns off file transfer and remote log-in Internet access.
- **enable** — Turns on file transfer and remote log-in Internet access.

The default value is **disable** if you choose not to install the Gateway during the DECnet–ULTRIX installation procedure. Otherwise, the default value is **enable**.

gateway user *login-name*
Specifies a default log-in name under which the DECnet objects **fal** and **dlogind** are to run if gateway access is enabled and a gateway request is received. The range is 1 through 32 characters.

identification *id-string*
Specifies the text identification string for the executor node. The range is 1 through 32 characters. You must use double quotation marks (") to delimit any string containing blanks or tabs. To include a quoted string within an identification string, enclose it within single quotation marks to distinguish the quotes from a string delimiter.

inactivity timer *seconds*
Specifies the maximum time the executor allows a link to remain idle (no user data traffic) before it checks to see whether the circuit still works. The range is 1 through 1024.

incoming proxy
Specifies whether the executor honors or ignores proxy log-in requests present on incoming logical links. Choose one of the following modes:

- **disable** — All incoming proxy requests are ignored. Instead, access control information supplied in each connect request is used to validate the request.
- **enable** — (Default.) Incoming proxy requests are honored, based on the source user, the source node, and the access control information.

incoming timer *seconds*
Specifies the maximum time a process has to answer an incoming connect request. If the process does not answer the connect request within this time interval, the node rejects the connect request on behalf of the process. The range is 1 through 1024.

maximum links *number*
Specifies the maximum number of active logical links for the executor node. The range is 1 through 1024. If you use this parameter in a **set** command, you must specify a value greater than the current maximum number of links. To decrease the value, you must issue a **define** command and restart the system.

maximum node counters *number*
Specifies the maximum number of node counters allowed. The range is 4 through 255, and the default is 32.

If you use this parameter in a **set** command, you must specify a value that is greater than the current maximum number of node counters. To decrease the value, you must issue a **define** command and restart the system.

name *node-name*
Specifies the node name of the executor.

outgoing proxy
Specifies whether or not the executor requests proxy log-in on outgoing connect requests. Choose one of the following modes:

- **disable** — The executor does not request proxy log-in on outgoing logical links, even when the user process attempts to enable it.
- **enable** — (Default.) The executor requests proxy log-in on all outgoing logical links, unless the user process explicitly disables it.

outgoing timer *seconds*
Specifies the maximum time the executor node waits for a pending connect request to be answered at a destination node. If the request is not answered in this time interval, the source process receives an error indication. The range is 1 through 1024.

pipeline quota *number*
Specifies the maximum number of bytes of buffer space that NSP can use for transmission and reception for each logical link. The range is 2,000 to 16,000. For satellite communications, you should use a value of 6,000 bytes or greater. The default is 4,096.

retransmit factor *number*
Specifies the number of times the executor restarts the retransmission timer before the logical link is disconnected. The range is 1 through 1024.

segment buffer size *number*
Specifies the size of the NSP message segment to be sent. This value is the maximum size message that the End Communications layer can transmit; it does not include routing or data link overhead. The range is 1 through 1,478, and the default is 576.

EXAMPLES

1. This command defines the executor address to 4.21 in the permanent database the next time the network is started up.

```
ncp>define executor address 4.21 RET
```

2. This command defines the maximum number of active logical links for the executor node to 20 in the volatile database.

```
ncp>define executor maximum links 20 RET
```

define line

DESCRIPTION

Modifies specified line parameters in the permanent database.

RESTRICTION

When you change the protocol and duplex parameter values of a point-to-point device, the operation mode of the device does not change until you restart the circuit. Therefore, you should turn the circuit off before changing the parameter, and then turn it back on afterward. For example:

```
ncp>define circuit dmv-0 state off RET
ncp>define line dmv-0 protocol ddcmp dmc duplex half RET
ncp>define circuit dmv-0 state on RET
```

SYNTAX

define line *line-id* { **controller** { **loopback** | **normal** } | **duplex** { **full** | **half** } | **protocol** { **ddcmp dmc** | **ddcmp point** | **ethernet** } }

where

line *line-id*
: Specifies the line for which parameters are to be modified.

controller
: Specifies the controller mode for the line. Choose one of the following modes:

  loopback — Internal device loopback mode.

  normal — Normal operating mode.

duplex
: Represents the Physical Link hardware duplex mode of the line device. (Valid for DDCMP point-to-point devices only.) Choose one of the following modes:

  full — Full-duplex mode.

  half — Half-duplex mode.

protocol
: Specifies the Data Link protocol to be used on the line. Choose one of the following modes:

  ddcmp dmc — Protocol for the DMC emulator. This setting is valid for dmv lines only.

  ddcmp point — Protocol for one end of a DDCMP point-to-point connection. This is the default for all DDCMP point-to-point connections, including dmv, dmc, and dmr lines.

  ethernet — Protocol for an Ethernet line. This is the default and the only valid choice for Ethernet lines (una, qna, sva, and bnt).

EXAMPLE

This command sets the controller for line una-0 to the **loopback** state.

```
ncp>define line una-0 controller loopback RET
```

define logging

DESCRIPTION

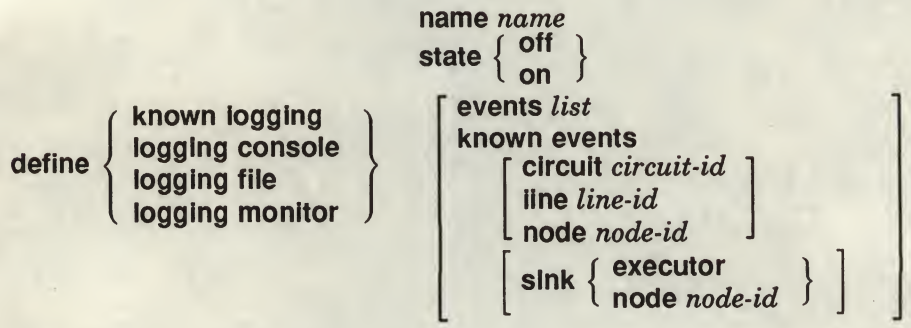
Creates or modifies logging component parameters in the permanent database.

RESTRICTIONS

You must have superuser privileges to use this command.

Whenever you specify a *circuit*, *line*, *node*, or *sink* in a **define logging** command, you must also specify an **events** *list* or **known events** parameter.

SYNTAX



where

circuit *circuit-id*
Logs the specified event(s) occurring on the specified circuit.

events *list*
Specifies the event class and type(s) to be logged.

known events
Specifies that all events that DECnet–ULTRIX can generate are to be logged.

known logging
Indicates that the specified parameters are to be created or modified for all known logging components.

line *line-id*
Logs the specified event(s) occurring on the specified line.

logging console
Indicates that the specified parameters are to be created or modified for the console logging component.

logging file
Indicates that the specified parameters are to be created or modified for the file logging component.

logging monitor
Indicates that the specified parameters are to be created or modified for the monitor logging component.

name *name*
Specifies the name of the console, monitor program, or file to which events are to be logged. The default name for a console is **/device/console**; for a monitor program it is **evi** and for a file it is **/usr/adm/eventlog**.

node *node-id*
Logs the specified event(s) occurring on the specified node.

sink	Identifies the node on which the specified events are to be logged. Choose one of the following qualifiers:
	executor (Default) executor node.
	node *node-id* The specified remote node.
state	Sets the operational state of the logging component on the executor node. When the state is **off**, events are discarded.

EXAMPLES

1. This command directs any known events on circuit una-0 to node BOSTON.

```
ncp>define known logging known events circuit una-0
sink node boston RET
```

2. This command directs any occurrence of event 2.1 to the console at node PARIS.

```
ncp>define logging console event 2.1 sink node paris RET
```

define node

DESCRIPTION

Adds or modifies a node specification in the permanent database.

RESTRICTION

You must have superuser privileges to execute this command.

SYNTAX

define node *node-id* { **address** *node-address* | **diagnostic file** *file* | **dump file** *file* | **hardware address** *E-address* | **host** *node-id* | **load file** *file* | **name** *node-name* | **secondary [loader]** *file* | **service** { **disable** | **enable** } | **service circuit** *circuit-id* | **[service] password** *service-password* | **tertiary [loader]** *file* }

where

address *node-address*
: Specifies a new (unassigned) node address to be associated with the node-name used as the *node-id* in this command.

diagnostic file *file*
: (For Ethernet nodes only.) Specifies the file to be read when the node is down-line loaded and requests diagnostics.

dump file *file*
: Specifies the file that is to receive a copy of the system at the time of the crash when the node is up-line dumped.

hardware address *E-address*
: Identifies the Ethernet address that was originally assigned to the Ethernet controller for the node. This address is used during operations such as down-line load to communicate with the system before it has set up its physical address.

host *node-id*
: Specifies a host node for all service operations. The default value is the executor node ID.

load file *file*
: Specifies a file containing the system software for down-line loading to a node.

name *node-name*
: Specifies a new (unassigned) node name to be associated with the *node-address* used as the *node-id* in this command.

node *node-id*
: Specifies the node for which the specified function is to be performed.

secondary [loader] *file*
: Specifies a file containing secondary loader software for down-line loading to the node.

service	Specifies whether the node is enabled or disabled for down-line loading.
service circuit *circuit-id*	Specifies the circuit to be used for down-line loading and up-line dumping. This circuit is the default value for the **via** parameter of the **load** command.
[service] password *service-password*	Specifies the password required to trigger the bootstrap mechanism on the node. The password is a hexadecimal value of 1 to 16 characters.
tertiary [loader] *file*	Specifies a file containing tertiary loader software for down-line loading to the node.

EXAMPLE

This command associates the name BURGER with node 12.

```
ncp>define node 12 name burger RET
```

define object

DESCRIPTION

Creates or modifies parameters for specified object(s) in the permanent database.

RESTRICTION

You must have superuser privileges to execute this command.

SYNTAX

define { **object** *object-name* | **known objects** } { **accept** { **deferred** | **immediate** } | **default user** *login-name* | **file** *file-id* | **number** *number* | **type** { **sequenced packet** | **stream** } }

where

accept
Specifies whether incoming connect requests are to be accepted or rejected by the DECnet object spawner or whether they are to be forwarded to the object for processing. Choose one of the following modes:

- **deferred** — The object processes any optional data sent with a connect request and can set optional data to be returned when it accepts or rejects the connect.
- **immediate** — (Default.) The DECnet object spawner determines whether to accept or reject all connect requests.

default user *login-name*
Specifies a default log-in name under which the object is to run if no access control information is supplied with a connect request and no proxy information is defined locally.

file *file-id*
Specifies an executable file or shell script used to invoke the specified object.

known objects
Specifies that parameters are to be created or modified for all known objects.

number *number*
Specifies the object number to be identified with the specified object name. The default is 0. See Appendix B for a list of object names and numbers.

object *object-name*
Specifies that parameters are to be created or modified for the named object only (a maximum of 16 alphanumeric characters).

type	Identifies the socket type. Choose one of the socket types:	
	sequenced packet	(Default) provides a bidirectional, reliable, sequenced, and unduplicated flow of data while preserving record boundaries.
	stream	provides same data flow properties as above without record boundaries.

EXAMPLE

This command defines the Network Management Listener (**nml**) as object number 19 with **/etc/nml** as the executable file. It also specifies that **nml** is to run under the log-in name "guest" if no access control information is supplied with a connect request and no proxy information is defined locally.

```
ncp>define object nml number 19 file /etc/nml default user guest RET
```

help

DESCRIPTION

Displays general information about **ncp** commands and parameters. Typing **help** displays the topics for which information is available; typing help and a specific topic or command name displays information about that topic or command.

SYNTAX

help [*topic*...]

topic — A command word listed in the **help** display. You may specify up to eight topics separated by spaces or tabs.

EXAMPLES

1. This command displays all command verbs for which further information exists.

```
ncp>help RET

Help available for:
clear           command         define          exit
help            list            load            loop
parameters      prompting       purge           set
show            tell            trigger         zero
ncp>
```

2. This command provides a description of the **ncp** command **clear circuit** and displays command words for which further information exists.

```
ncp>help clear circuit RET

clear circuit

Use the clear circuit command to remove circuit parameters from the
volatile database on the executor node.  Use the purge circuit
command to remove circuit parameters from the permanent database
on the executor node.

clear    circuit circuit-id     (parameters...)

Help available for:
circuit-id          all          receive password     transmit password
examples
ncp>
```

3. This command provides a description of the **ncp** command **show** and displays command words for which further information exists.

```
ncp>help show RET

show

Use the show command to display information from the volatile database
on the executor node.  Use the list command to display information
from the permanent database on the executor node.
```

```
Help available for:
characteristics      counters      events      status
summary              known         circuit     executor
line                 logging       node        object
module
ncp>
```

list circuit

DESCRIPTION

Displays specified circuit information stored in the permanent database.

SYNTAX

list { **circuit** *circuit-id* | **known circuits** } [**characteristics** | **status** | **summary**]

where

characteristics	Displays parameters that are currently set for the circuit.
circuit *circuit-id*	Displays information for the specified circuit only.
known circuits	Displays information for all known circuits.
status	Shows information that usually reflects network activity for the running network. Depending on the component, this can include the local node and its operational state, reachable and unreachable nodes, and circuits and lines and their operational states.
summary	(Default) shows abbreviated information provided for the **characteristics** and **status** display types.

EXAMPLE

This command displays circuit characteristics for all known circuits in the permanent database.

```
ncp>list known circuits characteristics RET

Known Circuit Permanent Characteristics as of Tue Nov 21 10:57:23 EST 199(

Circuit  =  UNA-0

Hello timer                          = 10
Type                                 = Ethernet

ncp>
```

list executor

DESCRIPTION

Displays specified local node information stored in the permanent database.

SYNTAX

list executor [**characteristics** | **status** | **summary**]

where

characteristics — Displays parameters that are currently set for the executor.

status — Shows information that usually reflects network activity for the running network. Depending on the component, this can include the local node and its operational state, reachable and unreachable nodes, and circuits and lines and their operational states.

summary — (Default) shows abbreviated information provided for the **characteristics** and **status** display types.

EXAMPLE

This command displays local node status information from the permanent database.

```
ncp>list executor status RET
Executor Permanent Status as of WED Nov 15 16:13:45 EST 1990
Executor node =  2.95 (OHIO)
State                                = On
ncp>
```

list line

DESCRIPTION

Displays specified line information stored in the permanent database.

SYNTAX

list { **line** *line-id* / **known lines** } [**characteristics** / **status** / **summary**]

where

line *line-id*	Displays information for the specified line only.
known lines	Displays information for all known lines.
characteristics	Displays parameters that are currently set for the line.
status	Shows information that usually reflects network activity for the running network. Depending on the component, this can include the local node and its operational state, reachable and unreachable nodes, and circuits and lines and their operational states.
summary	(Default) shows abbreviated information provided for the **characteristics** and **status** display types.

EXAMPLE

This command displays information about line una-0.

```
ncp>list line una-0 summary RET

Line Permanent Summary as of Wed Nov 15 16:17:06 EST 1990

Line                    State

BNT-0                   On
ncp>
```

list logging

DESCRIPTION

Displays specified logging information stored in the permanent database.

SYNTAX

list { **known logging** | **logging console** | **logging file** | **logging monitor** } [**characteristics** | **status** | **summary** | **events**] [**known sinks** | **sink node** *node-id*]

where

characteristics — Displays parameters that are currently set for the executor, line, or circuit.

events — Displays event class and type information for the given logging component. Choose one of the following qualifiers:

- **known sinks** — (Default) displays logging information for all known sink nodes.
- **sink node** *node-id* — Displays logging information for the specified sink node only.

known logging — Displays information for all known logging components.

logging console — Displays information for the console logging component.

logging file — Displays information for the file logging component.

logging monitor — Displays information for the monitor logging component.

status — Shows information that usually reflects network activity for the running network. Depending on the component, this can include the local node and its operational state, reachable and unreachable nodes, and circuits and lines and their operational states.

summary — (Default) shows abbreviated information provided for the **characteristics** and **status** display types.

EXAMPLE

This command displays event class and type information for the logging file on node N1834P.

```
ncp>list logging file events sink node n1834p RET
```

list node

DESCRIPTION

Displays specified node information stored in the permanent database.

SYNTAX

list { **node** *node-id* | **known nodes** } [**characteristics** | **status** | **summary**]

where

characteristics	Displays parameters that are currently set for the node.
known nodes	Displays information for all known nodes.
node *node-id*	Displays information for the specified node only.
status	Shows information that usually reflects network activity for the running network. Depending on the component, this can include the local node and its operational state, reachable and unreachable nodes, and circuits and lines and their operational states.
summary	(Default) shows abbreviated information provided for the **characteristics** and **status** display types.

EXAMPLE

This command displays error and performance statistics for all known nodes in the permanent database.

```
ncp>list node art summary RET

Node Permanent Summary as of Tue Nov 21 11:03:24 EST 1990

Executor node  =  2.39 (ART)

State                         = On
ncp>
```

list object

DESCRIPTION

Displays specified object information stored in the permanent database.

SYNTAX

list { **object** *object-name* / **known objects** } [**characteristics** / **summary**]

where

characteristics	Displays parameters that are currently set for the object.
counters	Provides counter information for circuits, lines, and nodes.
known objects	Displays information for all known objects.
object *object-name*	Displays information for the specified object only.
status	Shows information that usually reflects network activity for the running network. Depending on the component, this can include the local node and its operational state, reachable and unreachable nodes, and circuits and lines and their operational states.
summary	(Default) shows abbreviated information provided for the **characteristics** and **status** display types.

EXAMPLE

This command displays information about the Network Management Listener (**nml**).

```
ncp>list object nml RET

Object Permanent Summary as of Wed Nov 15 16:27:55 EST 1990

Object          Number          File

nml             19              /usr/ect/nml
ncp>
```

load node

DESCRIPTION

Down-line loads a specified remote node on the same Ethernet. Any parameter that you do not specify defaults to the value in the volatile database on the executor node.

With the **load node** command, the node that initiates the loading sequence is also the load host that performs the down-line load.

RESTRICTIONS

Before you can execute this command:

- You must have superuser privileges.
- The **mop_mom** utility must be installed during the the ULTRIX software installation.
- Service must be enabled on the remote node.
- You must have the service password if a DECnet service password is defined on the remote node.

SYNTAX

load node *node-id* { **address** *node-address* | **from** *load-file* | **host** *node-id* | **name** *node-name* | **physical address** *E-address* | **secondary [loader]** *file* | **[service] password** *service-password* | **tertiary [loader]** *file* | **via** *circuit-id* }

where

address *node-address*
: Specifies the address that the node is to use when it comes up.

from *load-file*
: Specifies the file containing the system software to be down-line loaded.

host *node-id*
: Specifies the default host that the node is to use when it comes up.

name *node-name*
: Specifies the name that the node is to use when it comes up.

node *node-id*
: Specifies the node to be down-line loaded.

physical address *E-address*
: (For Ethernet nodes only.) Identifies the Ethernet physical address that the node currently uses to identify itself. (Required for Ethernet circuits if the hardware address parameter has not been specified in the volatile database; see **set node**.)

secondary [loader] *file*	Specifies a file containing secondary loader software for down-line loading to the node.
[service] password *service password*	Specifies the password required to trigger the bootstrap mechanism on the node. The password is a hexadecimal value of 1 to 16 characters.
tertiary [loader] *file*	Specifies a file containing tertiary loader software for down-line loading to the node.
via *circuit-id*	Specifies the circuit over which the load is to take place.

EXAMPLE

This command loads node BOSTON using a service password.

```
ncp>load node boston service password FF55 RET
```

load via

DESCRIPTION

Down-line loads a remote node on the same Ethernet by way of the specified circuit. You must include the physical address in the command.

With the **load via** command, the node that initiates the loading sequence is also the load host that performs the down-line load.

RESTRICTIONS

Before you can execute this command:

- You must have superuser privileges.
- The **mop_mom** utility must be installed during the ULTRIX software installation.
- Service must be enabled on the remote node.
- You must have the service password if a DECnet service password is defined on the remote node.

SYNTAX

load via *circuit-id* { **address** *address* | **from** *load-file* | **host** *node-id* | **name** *node-name* | **physical address** *E-address* | **secondary [loader]** *file* | **[service] password** *service-password* | **tertiary [loader]** *file* }

where

address *node-address*
: Specifies the address that the node is to use when it comes up.

from *load-file*
: Specifies the file containing the system software to be down-line loaded.

host *node-id*
: Specifies the default host that the node is to use when it comes up.

name *node-name*
: Specifies the name that the node is to use when it comes up.

physical address *E-address*
: (For Ethernet nodes only.) Identifies the Ethernet physical address that the node currently uses to identify itself. If the circuit is an Ethernet circuit, you must include the physical address in the command.

secondary [loader] *file*
: Specifies a file containing secondary loader software for down-line loading to the node.

[service] password *service-password*	Specifies the password required to trigger the bootstrap mechanism on the node. The password is a hexadecimal value of 1 to 16 characters.
tertiary [loader] *file*	Specifies a file containing tertiary loader software for down-line loading to the node.
via *circuit-id*	Specifies the circuit over which the load is to take place.

EXAMPLE

This command loads the node connected to the executor node by circuit una-0.

```
ncp>load via una-0 physical address aa-00-03-00-01-19 service password FF55 RET
```

loop circuit

DESCRIPTION

Tests a circuit between the executor and another node in the network. You can specify a destination node using either its node name or its physical address. If you do not specify a destination node, the loop request is sent to the multicast address and the first node to respond completes the loop test.

If you specify a destination node, you can also specify a third node to assist with the test. The **help** parameter lets you specify the type of assistance you want.

RESTRICTION

You must have superuser privileges to execute this command.

SYNTAX

loop circuit [**node** *node-name* | **physical address** *E-address* [**help** *help-type* [{ **full** | **receive** | **transmit** } { **node** *node-name* | **assistant node** *node-name* | **physical address** *E-address* | **assistant physical address** *E-address* }]]]

[**count** *count* | **length** *length* | **with** { **mixed** | **ones** | **zeros** }]

where

assistant physical address *E-address*
: Specifies the physical address (not a multicast address) of an Ethernet node that is to assist in the loop circuit test.

assistant node *node-name*
: Specifies the name of an Ethernet node that is to assist in the loop circuit test.

circuit *circuit-id*
: (Does not apply to X.25 circuits.) Specifies the circuit to use for the loopback test.

count *count*
: Specifies the number of blocks to be sent during loopback testing. The valid range is 1 (default) through 1024.

help *help-type*

Specifies the degree to which a third node is to assist with an Ethernet loop circuit test. Choose one of the following types:

full (Default if an assistant node is specified but **help** is not.) The assistant node is to both receive and transmit the test packet (see the example).

receive The assistant node is only to receive the test packet.

transmit The assistant node is only to transmit the test packet.

NOTE

If you specify **help**, you must also specify either the **physical address** and **assistant physical address** parameters or, if the addresses are not known, the **node** and **assistant node** parameters.

length *length*

Specifies the length in bytes of each block to be sent during loopback testing. The default is 40 bytes. When testing over the Ethernet, the allowable length is from 1 byte to the maximum length of the data pattern, which varies according to the level of assistance:

Level of Assistance	Maximum Length
No assistance	1486 bytes
Transmit or receive assistance	1478 bytes
Full assistance	1470 bytes

node *node-name*

Specifies the name of an Ethernet node that is to be the destination of a loop-test message.

physical address *E-address*

Specifies the physical address (not a multicast address) of an Ethernet node that is to be the destination of a loop-test message.

with

Specifies the type of binary information to be sent during testing. If you omit this parameter, a combination of ones and zeros (the **mixed** data type) is sent. Choose one of the following data types:

mixed (Default.) A combination of ones and zeros.

ones

zeros

EXAMPLE

This command tests the circuit una-0 with the assistance of the node specified in assistant physical address by performing the following tasks:

1. The initiating node sends a test packet to the assistant node.
2. The assistant node processes the packet and passes the packet to the destination node specified in the physical address.
3. The destination node receives the packet and transmits the packet back to the assistant node.
4. The assistant node then returns the packet to the initiating node.
5. The **count** parameter indicates that this process is to be performed 10 times.

```
ncp>loop circuit una-0 help full physical address aa-00-04-00-f9-04
assistant physical address aa-00-04-00-04-a9 count 10 RET
```

loop executor

DESCRIPTION

Tests the executor node in the network. You can include access control information if the node requires it. This command causes test blocks of data to be transmitted to the specified node.

SYNTAX

loop executor [**count** *count* | **length** *length* | **with** { **mixed** | **ones** | **zeros** }]

where

executor — Specifies the executor node for loopback testing.

count *count* — Specifies the number of blocks to be sent during loopback testing. The valid range is 1 (default) through 1024.

length *length* — Specifies the length in bytes of each block to be sent during loopback testing. The length must be a decimal integer in the range of 1 through *n*, where *n* must be less than the smaller of either the local looper buffer size or the remote mirror buffer size. The default is 40 bytes.

node *node-id* — Specifies a node for loopback testing.

with — Specifies the type of binary information to be sent during testing. If you omit this parameter, a combination of ones and zeros (the **mixed** data type) is sent. Choose one of the following data types:

- **mixed** (Default.) A combination of ones and zeros.
- **ones**
- **zeros**

EXAMPLE

This command loops 10 blocks of mixed test messages to executor node. Each block is 40 bytes.

```
ncp>loop executor count 10 RET
```

loop node

DESCRIPTION

Tests a node in the network. You can include access control information if the node requires it. This command causes test blocks of data to be transmitted to the specified node.

SYNTAX

loop { **node** *node-id*[*acc-con-info*] } [**count** *count* | **length** *length* | **with** { **mixed** | **ones** | **zeros** }]

where

acc-con-info	Specifies access control information (if required by the remote node).
count *count*	Specifies the number of blocks to be sent during loopback testing. The valid range is 1 (default) through 1024.
length *length*	Specifies the length in bytes of each block to be sent during loopback testing. The length must be a decimal integer in the range of 1 through *n*, where *n* must be less than the smaller of either the local looper buffer size or the remote mirror buffer size. The default is 40 bytes.
node *node-id*	Specifies a node for loopback testing.
with	Specifies the type of binary information to be sent during testing. If you omit this parameter, a combination of ones and zeros (the **mixed** data type) is sent. Choose one of the following data types:
mixed	(Default.) A combination of ones and zeros.
ones	
zeros	

EXAMPLE

This command loops 10 blocks of mixed test messages to remote node OHIO. Each block is 40 bytes.

```
ncp>loop node ohio count 10 RET
```

purge circuit

DESCRIPTION

Removes specified circuit parameters from the permanent database.

RESTRICTION

You must have superuser privileges to execute this command.

SYNTAX

purge circuit *circuit-id* [**receive password** | **transmit password**]

where

circuit *circuit-id*	Specifies the circuit for which parameters are to be removed.
receive password	Removes the circuit's receive password.
transmit password	Removes the circuit's transmit password.

EXAMPLE

This command deletes all parameters for circuit una-0 from the permanent database.

```
ncp>purge circuit una-0 RET
```

purge executor

DESCRIPTION

Resets specified executor parameters to their initial values in the permanent database.

RESTRICTION

You must have superuser privileges to execute this command.

SYNTAX

purge executor { **identification** | **incoming timer** | **outgoing timer** }

where

identification	Removes the text identification string for the executor node. The valid range is 1 to 32 alphanumeric characters.
incoming timer	Resets the local node's incoming timer to its default value.
outgoing timer	Resets the local node's outgoing timer to its default value.

EXAMPLE

This command resets the local node's incoming timer to its default value in the volatile database.

```
ncp>purge executor incoming timer RET
```

purge logging

DESCRIPTION

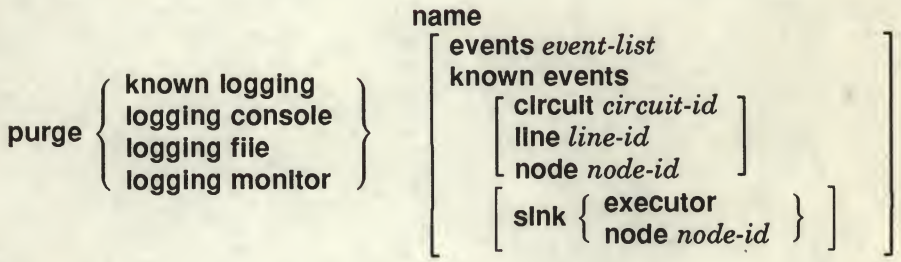
Removes specified logging parameters from the permanent database.

RESTRICTIONS

You must have superuser privileges to execute this command.

Whenever you specify a *circuit*, *line*, *node*, or *sink* in a **purge logging** command, you must also include an **event** *list* or **known events** parameter.

SYNTAX



where

circuit *circuit-id*
Inhibits event logging for the specified circuit.

events *event-list*
Removes the event class and types specified in *event-list* for the specified component.

known events
Removes all known events that DECnet–ULTRIX can generate for the specified component.

known logging
Removes parameters for all known logging components.

line *line-id*
Inhibits event logging for the specified line.

logging console
Removes parameters for the console-logging component.

logging file
Removes parameters for the file logging component.

logging monitor
Removes parameters for the monitor logging component.

name
Returns the specified logging component to its default name (console: **/device/console**; file: **/usr/adm/eventlog**; monitor **evl**).

node *node-id*
Inhibits event logging for the specified node.

sink
Inhibits logging of the specified events at the specified node. Choose one of the following parameters:

executor
This is the default setting. Events are not to be logged at the executor node.

node *node-id*
Events are not to be logged at the specified node

EXAMPLE

This command ceases logging of event 2.1 to the console.

```
ncp>purge logging console event 2.1 RET
```

purge node

DESCRIPTION

Removes the names associated with the nodes or removes specified node parameters from the permanent database.

RESTRICTION

You must have superuser privileges to execute this command.

SYNTAX

purge { **node** *node-id* / **known nodes** } { **diagnostic file** / **dump file** / **hardware address** / **host** / **load file** / **name** / **secondary loader** / **service circuit** / **service password** / **tertiary loader** }

or

purge { **node** *node-id* / **known nodes** } **all**

where

all	Removes all parameters for the specified node(s) so that the network no longer recognizes the node(s). Use of **all** precludes the use of any other parameters.
diagnostic file	Removes the identification of the down-line load diagnostic file.
dump file	Removes the up-line dump file identification.
hardware address	Removes the Ethernet address of the system hardware.
host	Removes the host node identification.
known nodes	Performs the specified function for all known nodes.
load file	Removes the identification of the down-line load file.
name	Removes the node name(s) associated with the specified node(s).
node *node-id*	Performs the specified function for the specified node only.
secondary loader	Removes the identification of the secondary down-line loading file.
service circuit	Removes the circuit parameter associated with the node for down-line loading purposes.
service password	Removes the password parameter. This password parameter is required to trigger the bootstrap mechanism of the node to be down-line loaded.

tertiary loader	**Removes the identification of the tertiary down-line loading file.**

EXAMPLE

This command removes all information for node BOSTON from the permanent database.

```
ncp>purge node boston all RET
```

purge object

DESCRIPTION

Removes specified object(s) from the permanent database.

RESTRICTION

You must have superuser privileges to execute this command.

SYNTAX

purge { **object** *object-name* | **known objects** }

where

known objects	Specifies that parameters are to be removed for all known objects.
object *object-name*	Specifies the object for which parameters are to be removed.

EXAMPLE

This command removes the network terminal handler **(dtermd)** from the permanent database.

```
ncp>purge object dtermd RET
```

set circuit

DESCRIPTION

Modifies specified circuit parameter(s) in the volatile database.

RESTRICTION

You must have superuser privileges to execute this command.

SYNTAX

set circuit *circuit-id* { **hello timer** *seconds* | **receive password** *password* | **state** { **off** | **on** } | **transmit password** *password* }

or

set circuit *circuit-id* **all**

where

all	Updates the volatile database with all of the parameters defined for the specified circuit in the permanent database. Use of **all** precludes use of any other parameters.
circuit *circuit-id*	Specifies the circuit for which parameters are to be modified.
hello timer *seconds*	Specifies the frequency of routing hello messages sent to adjacent nodes on the circuit. The range is 1 through 8,191.
receive password *password*	Specifies a 1- to 8-character ASCII password associated with the circuit that the executor expects to receive from the remote node during a routing initialization sequence.
transmit password *password*	Specifies a 1- to 8-character ASCII password associated with the circuit that the executor sends to the remote node during a routing initialization sequence.

EXAMPLES

1. This command updates the volatile database with all of the parameters defined for circuit qna-0 in the permanent database.

```
ncp>set circuit qna-0 all RET
```

2. This command makes circuit una-0 unavailable for use.

```
ncp>set circuit una-0 state off RET
```

set executor

DESCRIPTION

Creates or modifies specified executor node parameters in the volatile database.

NOTE

If you use the **set executor** command to issue a series of commands at a remote node, you can use the **tell** prefix to issue an **ncp** command to yet another remote node.

RESTRICTION

You must have superuser privileges to execute this command.

SYNTAX

set executor { **address** *node-address* | **delay factor** *number* | **delay weight** *number* | **gateway access** { **disable** | **enable** } | **gateway user** *login-name* | **identification** *id-string* | **inactivity timer** *seconds* | **incoming proxy** { **disable** | **enable** } | **incoming timer** *seconds* | **maximum links** *number* | **maximum node counters** *number* | **name** *node-name* | **outgoing proxy** { **disable** | **enable** } | **outgoing timer** *seconds* | **pipeline quota** *number* | **retransmit factor** *number* | **segment buffer size** *number* | **state** { **on** | **off** | **shut** | **restricted** } }

or

set executor all

where

all Updates the volatile database with all of the parameters defined for the executor node in the permanent database. Use of **all** precludes use of any other parameters.

delay factor *number*
Specifies a number to multiply times 1/16 of the estimated round-trip delay to a node in order to set the retransmission timer for that node. The range is 1 through 255.

delay weight *number*
Specifies a value to apply to a current round-trip delay to a remote node in order to update the estimated round-trip delay time. The range is 1 through 255.

gateway access
Specifies whether or not the executor allows the DECnet objects **fal** and **dlogind** to provide access to the Internet network. The default value is **disable** if you choose not to install the Gateway during the DECnet–ULTRIX installation procedure. Otherwise, the default value is **enable**. Choose one of the following values:

disable Turns off file transfer and remote log-in Internet access.

enable Turns on file transfer and remote log-in Internet access.

gateway user *login-name*
Specifies a default log-in name under which the DECnet objects **fal** and **dlogind** are to run if gateway access is enabled and a gateway request is received. The range is 1 through 32 characters.

identification *id-string*
Specifies a text identification string for the executor node. The range is 1 through 32 characters. You must use double quotation marks (") to delimit a string with blanks or tabs. To include a quoted string within an identification string, enclose the quoted string within single quotation marks.

inactivity timer *seconds*
Specifies the maximum time the executor will allow a link to remain idle (no user data traffic) before it checks to see whether the circuit still works. The range is 1 through 1024.

incoming proxy
Specifies whether the executor honors or ignores proxy log-in requests present on incoming logical links. Choose one of the following values:

disable All incoming proxy requests are ignored. Instead, access control information supplied in each connect request is used to validate the request.

enable (Default.) Incoming proxy requests are honored, based on the source user, the source node, and the access control information.

incoming timer *seconds*
Specifies the maximum time a process has to answer an incoming connect request. If the process does not answer the connect request within this time interval, the node will reject the connect request on behalf of the process. The range is 1 through 1024.

maximum links *number*
Specifies the maximum number of active logical links for the executor node. The range is 1 through 1024. If you use this parameter in a **set** command, you must specify a value greater than the current maximum number of links. To decrease the value, you must issue a **define** command and restart the system.

maximum node counters *number*

Specifies the maximum number of node counters allowed. The range is 4 through 255, and the default is 32.

NOTE

If you use this parameter in a **set** command, you must specify a value that is greater than the current maximum number of node counters. To decrease the value, you must issue a **define** command and restart the system.

name *node-name*

Specifies the node name of the executor.

outgoing proxy

Specifies whether or not the executor requests proxy log-in on outgoing connect requests. Choose one of the following values:

disable The executor does not request proxy log-in on outgoing logical links, even when the user process attempts to enable it.

enable (Default.) The executor requests proxy log-in on all outgoing logical links, unless the user process explicitly disables it.

outgoing timer *seconds*

Specifies the maximum time the executor node waits for a pending connect request to be answered at a destination node. If the request is not answered in this time interval, the source process receives an error indication. The range is 1 through 1024.

pipeline quota *number*

Specifies the maximum number of bytes of buffer space that NSP can use for transmission and reception for each logical link. The range is 2,000 to 16,000. For satellite communications, you should use a value of 6,000 bytes or greater. The default is 4,096.

retransmit factor *number*

Specifies the number of times the executor restarts the retransmission timer before the logical link is disconnected. The range is 1 through 1024.

segment buffer size *number*

Specifies the size of the NSP message segment to be sent. This value is the maximum size message that the End Communications layer can transmit; it does not include routing or data link overhead. The range is 1 through 1,478, and the default is 576.

state

Specifies the executor node's operational state. Use of **state** precludes use of any other parameters. Choose one of the following values:

on Allows logical links.

off Does not allow new links, terminates existing links, and stops route-through traffic.

restricted Does not allow new inbound links.

shut Does not allow new links but does not terminate existing links switches to the **off** state when all links have quit.

EXAMPLE

This command sets the maximum number of active logical links for the executor node to 20 in the volatile database.

```
ncp>set executor maximum links 20 RET
```

set executor node

DESCRIPTION

Sets the default executor as the specified remote node. This causes subsequent remotely executable **ncp** commands to be executed at the specified destination.

RESTRICTION

You cannot use the **tell** prefix with this command.

SYNTAX

set executor node *node-id*[*acc-con-info*]

where

acc-con-info	Specifies access control information (if required by the remote node).
node *node-id*	Specifies the remote node (by address, alias, or name) where subsequent **ncp** commands will be executed.

EXAMPLE

This command sets remote node EARTH (*user*: people; *password*: peace) to executor status. Future commands will be sent to node EARTH for execution.

```
ncp>set executor node earth/people/peace RET
```

set line

DESCRIPTION

Modifies the specified line parameter(s) in the volatile database.

RESTRICTION

When you change the protocol and duplex parameter values of a point-to-point device, the operation mode of the device does not change until you restart the circuit. Therefore, you should turn the circuit off before changing the parameter, and then turn it back on afterward. For example:

```
ncp>set circuit dmv-0 state off [RET]
ncp>set line dmv-0 protocol ddcmp dmc duplex half [RET]
ncp>set circuit dmv-0 state on [RET]
```

SYNTAX

set line *line-id* { **controller** { **loopback** | **normal** } | **duplex** { **full** | **half** } | **protocol** { **ddcmp dmc** | **ddcmp point** | **ethernet** } }

or

set line *line-id* **all**

where

all
Updates the volatile database with the parameters defined for the specified line in the permanent database.

controller
Specifies the controller mode for the line. Choose one of the following qualifiers:

- **loopback** — Internal device loopback mode.
- **normal** — Normal operating mode.

duplex
Represents the Physical Link hardware duplex mode of the line device. (Valid for DDCMP point-to-point devices only.) Choose one of the following qualifiers:

- **full** — Full-duplex mode.
- **half** — Half-duplex mode.

line *line-id*
Specifies the line for which parameters are to be modified.

protocol — Specifies the Data Link protocol to be used on the line. Choose one of the following parameters:

ddcmp dmc	Protocol for the DMC emulator. This setting is valid for dmv lines only.
ddcmp point	Protocol for one end of a DDCMP point-to-point connection. This is the default for all DDCMP point-to-point connections, including dmv, dmc, and dmr lines.
ethernet	Protocol for an Ethernet line. This is the default and the only valid choice for Ethernet lines (una, qna, sva, and bnt).

EXAMPLE

This command sets the controller for line una-0 to the **loopback** state.

```
ncp>set line una-0 controller loopback RET
```

set logging

DESCRIPTION

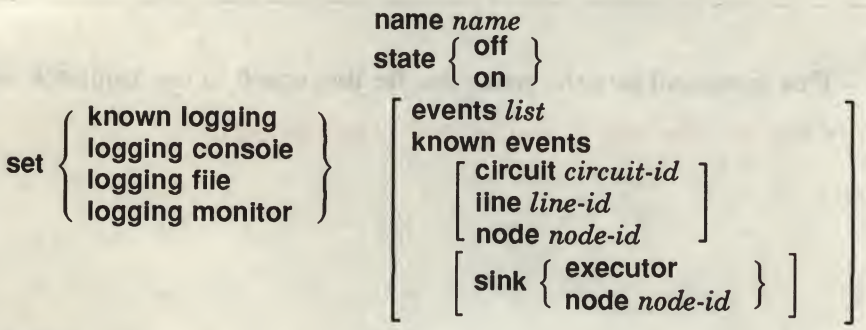
Creates or modifies logging component parameters in the volatile database.

RESTRICTIONS

You must have superuser privileges to use this command.

Whenever you specify a *circuit*, *line*, *node*, or *sink* in a **set logging** command, you must also include an **events** *list* or **known events** parameter.

SYNTAX



where

circuit *circuit-id*
Logs the specified event(s) occurring on the specified circuit.

events list
Specifies the event class and type(s) to be logged.

known events
Specifies that all events that DECnet–ULTRIX can generate are to be logged.

known logging
Indicates that the specified parameters are to be created or modified for all known logging components.

line *line-id*
Logs the specified event(s) occurring on the specified line.

logging console
Indicates that the specified parameters are to be created or modified for the console logging component.

logging file
Indicates that the specified parameters are to be created or modified for the file logging component.

logging monitor
Indicates that the specified parameters are to be created or modified for the monitor logging component.

name *name*
Specifies the name of the console, monitor program, or file to which events are to be logged. The default name for a console is **/device/console**; for a monitor program it is **evl** and for a file it is **/usr/adm/eventlog**.

node *node-id*
Logs the specified event(s) occurring on the specified node.

sink	Identifies the node where the specified events are to be logged. Choose one of the following qualifiers:
	executor (Default) executor node.
	node *node-id* The specified remote node.
state	Sets the operational state of the logging component on the executor node. When the state is **off**, events are discarded.

EXAMPLES

1. This command directs any known events on circuit una-0 to node BOSTON.

```
ncp>set known logging known events circuit una-0 sink node boston RET
```

2. This command directs any occurrence of event 2.1 to the console at node PARIS.

```
ncp>set logging console event 2.1 sink node paris RET
```

set node

DESCRIPTION

Modifies a node specification in the volatile database.

RESTRICTION

You must have superuser privileges to execute this command.

SYNTAX

set { **node** *node-id* } { **address** *node-address* | **diagnostic file** *file* | **dump file** *file* | **hardware address** *E-address* | **host** *node-id* | **load file** *file* | **name** *node-name* | **secondary [loader]** *file* | **service** { **disable** | **enable** } | **service circuit** *circuit-id* | **[service] password** *service-password* | **tertiary [loader]** *file* }

or

set { **node** *node-id* } **all**

where

all
Updates the volatile database with all of the parameters defined in the permanent database. Do not use any other parameter on the same command line.

address *node-address*
Specifies a new node address for the node.

diagnostic file *file*
(For Ethernet nodes only.) Specifies the file to be read when the node is down-line loaded and requests diagnostics.

dump file *file*
Specifies the file that is to receive a copy of the system at the time of the crash when the node is up-line dumped.

hardware address *E-address*
Identifies the Ethernet address that was originally assigned to the Ethernet controller for the node. This address is used during operations such as down-line load to communicate with the system before it has set up its physical address.

host *node-id*
Specifies a host node for all service operations. The default value is the executor node ID.

known nodes
(Valid for **all** only.) Specifies that the function is to be performed for all known nodes.

load file *file*	Specifies a file containing the system software for down-line loading to a node.
name *node-name*	Specifies a new (unassigned) node name to be associated with the *node-address* used as the *node-id* in this command.
node *node-id*	Specifies the node for which the specified function is to be performed.
secondary [loader] *file*	Specifies a file containing secondary loader software for down-line loading to the node.
service	Specifies whether the node is enabled or disabled for down-line loading.
service circuit *circuit-id*	Specifies the circuit to be used for down-line loading and up-line dumping. This circuit is the default value for the **via** parameter of the **load** command.
[service] password *service-password*	Specifies the password required to trigger the bootstrap mechanism on the node. The password is a hexadecimal value of 1 to 16 characters.
tertiary [loader] *file*	Specifies a file containing tertiary loader software for down-line loading to the node.

EXAMPLE

This command associates the name BURGER with node 12.

```
ncp>set node 12 name burger RET
```

set object

DESCRIPTION

Creates or modifies parameters for specified objects in the volatile database.

RESTRICTION

You must have superuser privileges to execute this command.

SYNTAX

set { **object** *object-name* / **known objects** } [**accept** { **deferred** / **immediate** } / **default user** *login-name* / **file** *file-id* / **number** *number* / **type** { **sequenced packet** / **stream** }]

or

set { **object** *object-name* / **known objects** } **all**

where

accept — Specifies whether incoming connect requests are to be accepted or rejected by the DECnet object spawner or whether they are to be forwarded to the object for processing. Choose one of the following modes:

- **deferred** — The object processes any optional data sent with a connect request and can set optional data to be returned when it accepts or rejects the connect.
- **immediate** — (Default.) The DECnet object spawner determines whether to accept or reject all connect requests.

all — Updates the volatile database with all of the parameters defined for the specified object in the permanent database. Use of **all** precludes use of any other parameters.

default user *login-name* — Specifies a default log-in name under which the object is to run if no access control information is supplied with a connect request and no proxy information is defined locally.

file *file-id* — Specifies an executable file or shell script used to invoke the specified object.

number *number* — Specifies the object number to be identified with the specified object name. The default is 0. See Appendix B for a list of object names and numbers.

known objects — Specifies that parameters are to be created or modified for all known objects.

object *object-name*	Specifies that parameters are to be created or modified for the named object only (a maximum of 16 alphanumeric characters).
type	Identifies the socket type. Choose one of the following socket types:
sequenced packet	(Default) provides a bidirectional, reliable, sequenced, and unduplicated flow of data while preserving record boundaries.
stream	Provides same data flow properties as above without record boundaries.

EXAMPLE

This command defines the Network Management Listener (**nml**) as object number 19 with **/etc/nml** as the executable file. It also specifies that **nml** is to run under the log-in name "guest" if no access control information is supplied with a connect request and no proxy information is defined locally.

```
ncp>set object nml number 19 file /etc/nml default user guest RET
```

show circuit

DESCRIPTION

Displays specified circuit information stored in the volatile database.

SYNTAX

show { **circuit** *circuit-id* | **known circuits** } [**characteristics** | **counters** | **status** | **summary**]

where

circuit *circuit-id*	Displays information for the specified circuit only.
characteristics	Displays parameters that are currently set for the circuit.
counters	Provides counter information for circuits, lines, and nodes.
known circuits	Displays information for all known circuits.
status	Shows information that usually reflects network activity for the running network. Depending on the component, this can include the local node and its operational state, reachable and unreachable nodes, and circuits and lines and their operational states.
summary	(Default) shows abbreviated information provided for the **characteristics** and **status** display types.

EXAMPLE

This command displays circuit error and performance statistics for all known circuits in the volatile database.

```
ncp>show known circuits counters RET

Known Circuit Volatile Counters as of Thur Nov 16 10:30:56 EST 1990

Circuit  =  una-0

          4127  Seconds since last zeroed
          6407  Terminating packets received
         11736  Originating packets sent
             0  Terminating congestions lost
             0  Transmit packets received
             0  Transmit packets sent
             0  Transmit congestion loss
             0  Initialization failure
       9679324  Bytes received
      61282059  Bytes sent
         75268  Data blocks received
         79489  Data blocks sent
             0  User buffer unavailable

ncp>
```

show executor

DESCRIPTION

Displays specified local node information stored in the volatile database.

SYNTAX

show executor [**characteristics** | **counters** | **status** | **summary**]

where

characteristics	Displays parameters that are currently set for the executor.
counters	Provides counter information for circuits, lines, and nodes.
status	Shows information that usually reflects network activity for the running network. Depending on the component, this can include the local node and its operational state, reachable and unreachable nodes, and circuits and lines and their operational states.
summary	(Default) shows abbreviated information provided for the **characteristics** and **status** display types.

EXAMPLE

This command displays local node status information from the volatile database.

```
ncp>show executor status RET

Executor Volatile Status as of Thu Nov 16 10:37:23 EST 1990

Executor node =  2.95 (OHIO)

State                                = On
Physical address                     = aa-04-00-00-53-10
ncp>
```

show line

DESCRIPTION

Displays specified line information stored in the volatile database.

SYNTAX

show { **line** *line-id* | **known lines** } [**characteristics** | **counters** | **status** | **summary**]

where

characteristics	Displays parameters that are currently set for the line.
counters	Provides counter information for circuits, lines, and nodes.
known lines	Displays information for all known lines.
line *line-id*	Displays information for the specified line only.
status	Shows information that usually reflects network activity for the running network. Depending on the component, this can include the local node and its operational state, reachable and unreachable nodes, and circuits and lines and their operational states.
summary	(Default) shows abbreviated information provided for the **characteristics** and **status** display types.

EXAMPLE

This command displays information about line una-0.

```
ncp>show line una-0 summary RET
Line Volatile Summary as of Thu Nov 16 10:40:17 EST 1990
Line                    State
UNA-0                   On
ncp>
```

show logging

DESCRIPTION

Displays specified logging information stored in the volatile database.

SYNTAX

show { **known logging** | **logging console** | **logging file** | **logging monitor** } [**characteristics** | **status** | **summary** | **events**] [**known sinks** | **sink node** *node-id*]

where

characteristics	Displays parameters that are currently set for the executor, line, or circuit.
events	Displays event class and type information for the given logging component.
known logging	Displays information for all known logging components.
known sinks	(Default) displays logging information for all known sink nodes.
logging console	Displays information for the console logging component.
logging file	Displays information for the file logging component.
logging monitor	Displays information for the monitor logging component.
sink node *node-id*	Displays logging information for the specified sink node.
status	Shows information that usually reflects network activity for the running network. Depending on the component, this can include the local node and its operational state, reachable and unreachable nodes, and circuits and lines and their operational states.
summary	(Default) shows abbreviated information provided for the **characteristics** and **status** display types.

EXAMPLE

This command displays event class and type information for the logging file on node N1834P.

```
ncp>show logging file events sink node n1834p RET
Logging Volatile Events as of Thu Nov 16 10:44:04 EST 1990
Logging = file
  No Information
ncp>
```

show node

DESCRIPTION

Displays specified node information stored in the volatile database.

RESTRICTION

No information is displayed for an end node until a link has been established to it. The node may appear to be unreachable even when it is not.

SYNTAX

show { **node** *node-id* | **active nodes** | **known nodes** } [**characteristics** | **counters** | **status** | **summary**]

where

active nodes	Displays information for all nodes that are adjacent, designated routers, or connected to the executor by a logical link.
characteristics	Displays parameters that are currently set for the node.
counters	Provides counter information for circuits, lines, and nodes.
known nodes	Displays information for all known nodes.
node *node-id*	Displays information for the specified node only.
status	Shows information that usually reflects network activity for the running network. Depending on the component, this can include the local node and its operational state, reachable and unreachable nodes, and circuits and lines and their operational states.
summary	(Default) shows abbreviated information provided for the **characteristics** and **status** display types.

EXAMPLE

This command displays error and performance statistics for all known nodes in the volatile database.

```
ncp>show node ohio counters RET

Node Volatile Counters as of Thu Nov 16 10:49:38 EST 1990

Executor node = 2.95 (OHIO)

                0  Aged packet loss
                0  Node unreachable packet loss
                0  Node out-of-range packet loss
                0  Oversized packet loss
                0  Packet format error
                0  Partial routing update loss
                0  Verification reject

ncp>
```

show object

DESCRIPTION

Displays specified object information stored in the volatile database.

SYNTAX

show { **object** *object-name* | **known objects** } [**characteristics** | **summary**]

where

characteristics	Displays parameters that are currently set for the object.
known objects	Displays information for all known objects.
object *object-name*	Displays information for the specified object only.
summary	(Default) shows abbreviated information provided for the **characteristics** and **status** display types.

EXAMPLE

This command displays information about the Network Management Listener (**nml**).

```
ncp>show object nml RET

Object Volatile Summary as of Thu Nov 16 10:53:46 EST 1990

Object              Number          File

nml                   19            /usr/etc/nml
ncp>
```

tell

DESCRIPTION

Sends an **ncp** command to a remote node for execution. **tell** sets the executor only for the command that it prefixes.

NOTE

If you use the **set executor** or **define executor** command to issue a series of commands at a remote node, you can still use the **tell** prefix to issue an **ncp** command to yet another remote node.

SYNTAX

tell *node-id* [*acc-con-info*] *ncp-command*

where

acc-con-info	Specifies access control information required by the remote node.
ncp *command*	Represents any **ncp** command that is remotely executable.
node-id	Specifies the node address or the node name of the remote node.

EXAMPLE

This command sends the **show executor** command to node ART, and tells ART to show characteristics of the executor node.

```
ncp>tell art show exec char RET
```

trigger node

DESCRIPTION

Initiates a down-line load to the specified remote node. Initiates the loading sequence for an unattended system.

NOTE

There is no way to determine the node that performs the down-line load.

RESTRICTIONS

Before you can execute this command:

- You must have superuser privileges.
- The **mop_mom** utility must be installed during the ULTRIX software installation.
- Service must be enabled on tne remote node.
- You must have the service password if a DECnet service password is defined on the remote node.

SYNTAX

trigger node *node-id* { **physical address** *E-address* | **[service] password** *hex-password* | **via** *circuit-id* }

where

node *node-id*	Specifies the remote node to be loaded.
physical address *E-address*	Identifies the Ethernet physical address of the remote node. Required for Ethernet circuits if the hardware address parameter has not been specified in the volatile database.
[service] password *hex-password*	Specifies the DECnet service password for maintenance operations such as down-line loading. Specify a hexadecimal value of 16 digits.
via *circuit-id*	Specifies the circuit over which the operation is to take place.

EXAMPLE

This command initiates a down-line load to node BOSTON, whose DECnet service password is aabb.

NOTE

Even though node BOSTON initiates the down-line load, there is no way to determine which host will actually perform the down-line load.

```
ncp>trigger node boston password aabb RET
```

trigger via

DESCRIPTION

Initiates a down-line load to the specified remote node through the specified circuit. Initiates the loading sequence for an unattended system through the specified circuit. The circuit identification is obtained from the volatile database on the executor node.

NOTE

There is no way to determine the node that performs the down-line load.

RESTRICTIONS

Before you can execute this command:

- You must have superuser privileges.
- The **mop_mom** utility must be installed during the ULTRIX software installation.
- Service must be enabled on the remote node.
- You must have the service password if a DECnet service password is disabled on the remote node.

SYNTAX

trigger via *circuit-id* { **physical address** *E-address* | **[service] password** *service-password* }

where

via *circuit-id*	Specifies the circuit over which the operation is to take place.
physical address *E-address*	(For Ethernet nodes only.) Identifies the Ethernet physical address that the node currently uses to identify itself. If the circuit is an Ethernet circuit, you must include the physical address in the command.
service password *service-password*	Specifies the remote node's service password.

EXAMPLE

This command initiates a down-line load sequence on the node connected to circuit una-1. The node's service password is ffaa.

```
ncp>trigger via una-1 physical address aa-00-03-00-01 password ffaa RET
```

zero circuit

DESCRIPTION

Sets circuit counters to zero for the specified circuit. The executor node maintains these counters for each circuit.

RESTRICTION

You must have superuser privileges to execute this command.

SYNTAX

zero circuit *circuit-id* [counters]

where

circuit circuit-id — Specifies the circuit for which counters are to be zeroed.

EXAMPLE

This command sets circuit counters to zero for circuit una-0.

```
ncp>zero circuit una-0 RET
```

zero executor

DESCRIPTION

Sets node counters associated with and maintained on the executor node to zero.

RESTRICTION

You must have superuser privileges to execute this command.

SYNTAX

zero executor [counters]

zero line

DESCRIPTION

Sets line counters to zero for the specified line. The executor node maintains these counters for each line.

RESTRICTION

You must have superuser privileges to execute this command.

SYNTAX

zero line *line-id* **[counters]**

where

line *line-id* — Specifies the line for which counters are to be zeroed.

EXAMPLE

This command sets the line counters to zero for line qna-0.

```
ncp>zero line qna-0  RET
```

zero node

DESCRIPTION

Sets node counters to zero for specified nodes in the volatile database. The executor node maintains node counters for each node.

RESTRICTION

You must have superuser privileges to execute this command.

SYNTAX

zero { **node** *node-id* / **known nodes** } **[counters]**

where

known nodes	Zeros counters for all known nodes.
node *node-id*	Zeros counters for the specified node.

EXAMPLE

This command sets the node counters to zero for node BOSTON.

```
ncp>zero node boston RET
```

Chapter 3

Error Messages

This chapter outlines the ncp error message format and lists all messages in alphabetical order, giving a short description of each. When possible, you should correct the error condition and retry the command.

3.1 ncp Error Message Format

The ncp error messages have the following format:

ncp [- Listener response]: *error message*[,*error detail*]
[*extra text* or *command echo*]

where

Listener response	Indicates when the error message is being displayed by the Network Management Listener.
error message	Is the reason for the failure.
error detail	Is a detailed explanation of the failure (for certain error messages.
extra text	Gives an additional system-specific explanation of the error condition.
command echo	Is the command in error (if applicable). An arrow points to the illegal condition.

The following example contains the error message, "Unrecognized keyword," and the command echo, "tell boston sho lines." This message indicates an error in the command line.

```
ncp: Unrecognized keyword
tell boston sho lines
```

The command should read, `"tell boston sho known lines."`

3.2 ncp Error Messages

This section lists the ncp error messages in alphabetical order, with a short description of each message.

Bad loopback response

The message returned in a loopback test does not match the message sent. This error is caused by a loopback protocol violation, bad data return, or bad message length return. Check the integrity of the line. Make sure that you are not on a noisy line.

Component in wrong state

The current operational state of the component precludes the requested operation. The error detail identifies the component type. Check to see that the component is in the correct state to perform the requested operation.

Connect failed

A logical link connect has failed for the reason described in one of the following error details:

Access control rejected

The remote node could not understand or would not accept the access control information. Make sure that you have a valid user name and password on the remote system; then try again.

Insufficient network resources

Either the local node or the remote node had insufficient network resources to create the logical link. You can increase the number of maximum links for DECnet–ULTRIX by using the **ncp set/define executor** command, or reduce the number of logical links in use.

Invalid node name format

The format of the specified node name is invalid. Use 1 to 6 alphanumeric characters, including at least one alphabetic.

Invalid object name format

The remote node did not understand the object name format used by **ncp** to identify the Network Management Listener. Contact the person responsible for your network.

Local node shutting down

The local node is shutting down and will not allow any logical link connections. Wait until your system resumes network activity, then try again to connect.

No response from object

The Network Management Listener did not respond; for example, it may have responded too slowly or terminated abnormally. Contact the person responsible for your network.

Node unreachable

No path exists to the remote node. Make sure that DECnet is installed on the remote node. Check to see that the DECnet circuit is running by using the **ncp show circuit status** command. Check the status of the remote node to see if it is up.

Object too busy

The remote nml object had insufficient resources available to accept the connect request. Contact the person responsible for your network.

Remote node shutting down

The remote node is shutting down and will not accept any logical link connections. Wait until the remote system resumes network activity, then try again to connect.

Unrecognized node name

The destination node name does not correspond to any known node address. Enter a valid node name, which consists of from 1 to 6 alphanumeric characters, including at least one alphabetic character. Check to see if the node name is defined in the DECnet database by using the ncp show node or show known nodes command. If the name is not defined, use the ncp set command to define the node.

Unrecognized object

The remote node does not have a Network Management Listener. Contact the person responsible for your network.

File I/O error

An error was encountered while reading or writing a file necessary to the requested operation. The error detail identifies the database where the error occurred. The file may be corrupted or may not exist. Make sure that the DECnet files copied to your system at installation are in the right directories. See *DECnet–ULTRIX Installation* for a list of the files.

File open error

A file necessary for the requested operation could not be opened. The error detail identifies the database where the error occurred. Make sure that the DECnet files copied to your system at installation are in the right directories. See *DECnet–ULTRIX Installation* for a list of the files.

Hardware failure

The hardware associated with the request could not perform the specified operation. Contact the person responsible for your network, or call your Field Service representative.

Incompatible management version

The Network Management Listener version is incompatible with ncp. You must go to the remote system to execute the command.

Invalid file contents

The requested operation could not be performed because the files contained data of an invalid form or value. The error detail identifies the database where the error occurred. Look at the file to see if it is corrupted. See *DECnet–ULTRIX Installation* for a list of the files.

Invalid identification

The identification of the component specified in the requested operation did not have the proper syntax. For example, a device name that should have the syntax dev-*n*, appears as **unas**. The error detail identifies the component type. Make sure that the component name and unit number, if applicable, are correct and configured into the system.

If this message is received on a **loop circuit** command, the following error detail may be included:

Unable to find device

The device specified by the circuit ID does not exist.

Invalid message format

The information sent by **ncp** to a Network Management Listener was improperly formatted or contained an invalid value. Submit an error report to your Digital Software Services representative.

Invalid parameter grouping

The parameters furnished by the user for the requested operation cannot be included in the same command. Check the appropriate **ncp** command description in this manual. Often, use of the **all** parameter in an **ncp** command precludes the use of any other command parameters.

Invalid parameter value

The value of a parameter furnished by the user for the requested operation was not acceptable (for example, a numeric parameter was out of range). The error detail identifies the type of parameter. Check the appropriate **ncp** command description in this manual, and retry the command using a different parameter value. If this message refers to the **length** parameter in a **loop command**, one of the following error details may be included. In each of these cases, the length was more than could be handled, and the maximum length is included with the error message. Retry the command using a shorter length.

Ethernet message size exceeded

The length specified for blocks to be looped exceeds the maximum allowed. (See the **loop** command descriptions in this manual for details.)

Looper size exceeded

The requested length exceeds the buffering capability of the active looper task.

Mirror size exceeded

The requested length exceeds the buffering capability of the network management loopback mirror.

Line communication error

The requested operation failed because of communication errors on the involved line. Make sure that your system is properly attached to the network and that the other node is up and running.

Line protocol error

The requested operation failed because of protocol errors on the involved line. This condition usually implies either incompatible line protocols or protocol-programming errors. It can also be caused by a line hardware error that was not detected by the line protocol. (Line protocol can mean either the Data Link Protocol or the Service Operation Protocol.)

Management program error

The network management software has detected an internal error. Contact the person responsible for your network and/or submit a report to your Digital Software Services representative.

If this error occurs on a loop command, one of the following error details may be provided:

Bad data pattern developed

The software is unable to build a message because of a program error.

Incorrect optional data size on accept

The optional accept data from mir was improperly formatted.

Unable to get physical address from device

A request for a device's Ethernet physical address failed.

Mirror connect failed

The logical link to the network management loopback mirror could not be connected. This error message usually has one of the following error details:

Abort by management

The connection was aborted by a third party, not by the user programs at either end of the connection. Contact the person responsible for your network.

Abort by object

A programming error in the network management loopback mirror caused it to abort the logical link. Submit an error report to your Digital Software Services representative.

Access control rejected

Either the remote node or the network management loopback mirror could not understand or would not accept the access control information. Make sure you have a valid user name and password on the remote node; then try again.

Connection rejected by object

The logical link could not be connected because the network management loopback mirror rejected the connection. This condition usually implies that the loopback mirror is too busy to accept another logical link. Try to connect later.

Disconnect by object

A programming error in the network management loopback mirror caused it to disconnect the logical link. Submit an error report to your Digital Software Services representative.

Insufficient network resources

Either the local node or the remote node had insufficient network resources to connect the logical link. You can increase the number of maximum links for DECnet–ULTRIX by using the **ncp set executor** and **define executor** commands, or reduce the number of links in use.

Local node shutting down

The executor node is in the process of shutting down and is not accepting any more logical link connections. Wait until your system resumes network activity, and then try again to connect.

No response from object

The network management loopback mirror did not respond; for example, it may have responded too slowly or terminated abnormally. Contact the person responsible for your network.

Node unreachable

No path exists to the remote node. Make sure that DECnet is installed on the remote node. Check to see that the DECnet circuit is running by using the **ncp show circuit status** command. Check the status of the remote node to see if it is up.

Object too busy

The remote **nml** object had insufficient resources available to accept the connect request. Contact the person responsible for your network.

Remote node shutting down

The remote node is shutting down and will not accept any logical link connections. Wait until the remote node resumes network activity, then try again to connect.

Unrecognized node name

The destination node name does not correspond to any known node address. Enter a valid node name, which consists of from 1 to 6 alphanumeric characters, including at least one alphabetic character. If the name is not defined in the DECnet database, use the **ncp set** command to define the node.

Unrecognized object

The remote node does not have a network management loopback mirror. Contact the person responsible for your network.

Mirror link disconnected

The logical link from ncp to the network management loopback listener was unexpectedly disconnected. This error message usually has one of the error details listed under "Mirror connect failed." See the error detail for a description of the error and a recommended action.

No room for new entry

The requested operation could not be performed because it required the addition of a new entry in a database that was already full. Contact the person responsible for the remote system.

Operation failure

The requested operation failed. If an error detail is not provided, see the network documentation for the remote system.

Oversized management command message

The ncp command message was too big to be received by the Network Management Listener. Submit an error report to your Digital Software Services representative.

Oversized management response

The message returned by the Network Management Listener was too big to be received by ncp. Submit an error report to your Digital Software Services representative.

Parameter missing

A necessary parameter for the requested operation was omitted. The error detail identifies the type of parameter. Check the appropriate ncp command description in this manual and reenter the command including the necessary parameter.

Parameter not applicable

The user supplied a parameter that is not applicable to the requested operation on the specified component. The error detail identifies the type of parameter. Check the appropriate ncp command description and reenter the command minus the problem parameter.

Parameter value out of range

A numeric parameter value is outside the allowable range. Identify the parameter by referring to the appropriate ncp command description; then reenter the command.

Parameter value too long

The parameter value was too long to be accepted by the Network Management Listener. The error detail identifies the type of parameter. Shorten the parameter value, and then reenter the command.

Privilege violation

The user does not have sufficient privilege to perform the requested operation. Log in as superuser and reenter the command, or contact the person responsible for your network.

Redundant parameter

A parameter has been entered twice in the same command. Eliminate the extra parameter and reenter the command.

Resource error

Network management had insufficient internal resources to perform the requested operation.

On a **loop** command, the following error detail may also be provided:

Unable to check loopback state

The software could not open a socket to see whether the device is in loopback state. Check to see that **dli** is configured into the system.

System-specific management function not supported

The requested operation is ULTRIX system-specific and is not supported by the Network Management Listener. You must perform this operation on the remote node.

Unrecognized command

The Network Control Program (**ncp**) does not support the command entered by the user. Refer to the **ncp** command descriptions in this manual.

Unrecognized component

The component specified by the user does not exist. The error detail identifies the component type. Make sure that the DECnet files copied to your system at installation are in the right directories. See *DECnet–ULTRIX Installation* for a list of the files.

Unrecognized function or option

The requested operation is not implemented by the executor. Refer to the **ncp** command descriptions in this manual.

Unrecognized keyword

One of the keywords in a command is unknown to **ncp**. The command echo flags the unrecognized keyword. Refer to the **ncp** command descriptions in this manual.

Unrecognized parameter type

The command parameter identified in the error detail is not implemented by the executor. Refer to the **ncp** command descriptions in this manual.

Unrecognized value

A parameter value specified by the user is unknown to **ncp**. The command echo flags the faulty parameter value. Refer to the relevant **ncp** command description in this manual.

Chapter 4

Event Messages

The DECnet–ULTRIX Event Logger (evl) is a network management tool that records network activity. The Event Logger can record two categories of event messages: event classes and event types. Event classes relate to specific layers of the DECnet architecture, while event types relate to specific events within an event class.

4.1 Event Classes

DECnet logging events fall into the event classes listed in Table 4-1. Events not logged by DECnet-ULTRIX may be logged by other remote nodes. Check the documentation for the remote system for details.

Table 4–1: Event Classes

Event Class	Description
0	Network Management layer
1[1]	Applications layer
2	Session Control layer
3	End Communications layer
4	Routing layer
5[1]	Data Link layer
6[1]	Physical Link layer
7-479[1]	Reserved for other DECnet products
480-511	Reserved for customer use

[1]Event class not logged by DECnet-ULTRIX.

4.2 Event Message Format

Event messages have the following format:

Event type *class.type*, [*event-text*]

Occurred *dd-mon-yy hh:mm:ss.s* on node *address* [(*node-name*)]

[*component-type component-name*]

[*data*]

where

class	Is the class in which the event occurred (see Table D-1.)
type	Is the specific event type number within that class.
event-text	Is the text describing the event (see Sections D.3 through D.6.)
dd-mon-yy	Is the date (day, month, and year) on which the event occurred.
hh:mm:ss.s	Is the time (hour, minutes, and seconds) at which the event occurred.
address	Is the address of the node at which the event occurred.
node-name	Is the name of the node at which the event occurred.
component-type	Is either line, circuit, or node. If an event is not associated with a particular component, this line is not present.
component-name	Is the name of the component that caused the event.
data	Is event-dependent text that gives more information about the event. Often this text includes the name or address for the component type for which the event applies. It may also provide additional information about the cause of the event.

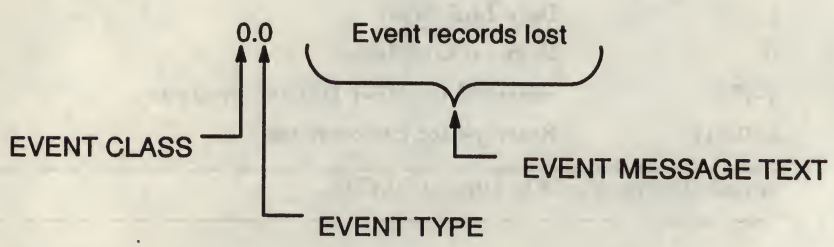
EXAMPLE:

This example shows an event-logging message indicating that an adjacent node is unavailable.

```
Event type 4.18, Adjacency down
Occurred 21-May-87 08:17:11.0 on node 19.12 (PITSBG)
Circuit UNA-0
Adjacency listener receive timeout
Adjacent node = 19.160
```

The following sections list DECnet–ULTRIX event messages by class and type for each layer. Figure D-1 shows the format in which the messages appear:

Figure 4–1: Event Message Format



4.3 Network Management Layer Events

Event	Message
0.0	**Event records lost** Events occurred too rapidly for the event logger to buffer them. This message does not display any event qualifiers.
0.6	**Passive loopback** The software initiated or terminated a passive loopback test on behalf of an adjacent node. This message displays the circuit name to which the event applies and the state of operation qualifier (initiated or terminated).

4.4 Session Control Layer Events

Event	Message
2.0	**Local node state change** The operational state of the local node changed because of an operator command. Note that the transition from **shut** to **off** also happens automatically when the last logical link is disconnected (under normal operation). This message displays the reason for the state change (operator command or normal operation), the old state (**on**, **off**, **shut**, or **restricted**), and the new state (**on**, **off**, **shut**, or **restricted**).
2.1	**Access control failure** The local node rejected a connect request because of invalid access control information. This message displays the name and address of the source node, the object type number and process ID of the source process requesting the connection, the object type number and process ID of the destination process to receive the connect request, and the invalid access control information (user ID, password, and account information).

4.5 End Communications Layer Event

Event	Message
3.2	**Node data base reused** The local node received a connect request from or tried to initiate an outgoing connect to a node for which there is no counter block. All counter blocks have been used, and one of the previously used blocks is available for this new node. This results in the loss of node counters for the node that formerly occupied the database entry. This message displays the address and name of the node for which the database entry was formerly used and the counters for that node.

4.6 Routing Layer Events

Event	Message
4.3	**Oversized packet loss** A packet has been discarded because it was too large to forward to the appropriate adjacent node. Normally, this condition occurs because the adjacent node's buffer is too small or the source node sent a packet that was too large. The latter condition can be handled by setting a smaller segment size at the source node. This message displays the packet header and the name of the circuit over which the packet was to be forwarded. For contents of the packet header, refer to the *DECnet Digital Network Architecture (Phase IV), Network Management Functional Specification*.
4.4	**Packet format error** A packet has been discarded because of a format error in the packet header. Usually, this results from a programming error in packet formatting by the adjacent node. It could also result from a circuit error not detected by circuit protocol. This message displays the first six bytes of the packet (in hexadecimal) and the name of the circuit to which the event applies. For a point-to-point line this message also displays the adjacent node.
4.6	**Verification reject** This message is displayed for point-to-point lines only. It displays the first six bytes of the packet header (in hexadecimal), the name of the circuit, and the adjacent node to which the packet applies. An invalid verification message was received. The password from the remote node does not match the 'circuit receive password' set up in the database.
4.8	**Circuit down** Under normal operating conditions, this event occurs when you set the circuit or executor to the **off** state. It also occurs if there is a hardware problem with the line or device. For point-to-point lines, this event can occur if the node listen timer expires or invalid data is received in the hello message. This message displays the name of the circuit to which the event applies. For point-to-point lines, this message also displays the adjacent node name. In the case of a hardware error, it also displays the following message: **Adjacent node listener received invalid data**
4.9	**Circuit down — operator initiated** This event is logged when the node identification in the hello message from the remote node is not the expected one. This message displays the name of the circuit and adjacent node to which the event applies. This message is displayed for point-to-point lines only.
4.10	**Circuit up** The basic initialization of a remote node has been completed by the device and transceiver. This message displays the name of the circuit to which the event applies, as well as the name and address of the newly initialized node. For a point-to-point line this message also displays the adjacent node.
4.11	**Initialization failure — circuit fault** This event is logged when a verification message is not received within the timeout period. This message is displayed for point-to-point lines only. It displays the circuit, adjacent node, and reason for failure.

Event	Message
4.12	**Initialization failure — software** This message is logged when the local node is unable to send the verification message requested by the remote node due to lack of system resources. This message is displayed for point-to-point lines only. It displays the circuit, adjacent node, packet header, and reason for failure.
4.13	**Initialization failure — operator initiated** The routing layer logs this message when it detects an area mismatch or a version skew during the circuit initialization sequence. This message is displayed for point-to-point lines only. It displays the circuit, adjacent node, packet header, and reason for failure.
4.15	**Adjacency up** For broadcast circuits (UNA and QNA), initialization has occurred with another node on the Ethernet. End nodes log this message for only one node. This message displays the adjacent node number.
4.18	**Adjacency down** The remote node has recycled. This event could result from a remote node restart or an invalid protocol message. The message displays the reason why the adjacent node is down.

4.7 Event Log Summary

Table 4-2 summarizes the events logged by the event logger:

Table 4–2: Event Log Summary

Class	Type	Entity[1]	Standard text	Counters
0	0	none	Event records lost	none
0	1	node	Automatic node counters	Node counters
0	5	node	Node counters zeroed	Node counters
0	8	any	Automatic counters	Counters
0	9	any	Counters zeroed	Counters
2	1	none	Access control reject	Source node, Source process, Destination process, User, Password, Account
3	0	none	Invalid message	Message, Source node
3	1	none	Invalid flow control	Message, Source node, Current flow control
3	2	node	Data base reused	NSP node counters
4	0	none	Aged packet loss	Packet header
4	1	circuit	Node unreachable, packet loss	Packet header, Adjacent node

[1]In this context, entity refers to a component or software module that can generate events.

(continued on next page)

Table 4–2 (Cont.): Event Log Summary

Class	Type	Entity[1]	Standard text	Counters
4	2	circuit	Node out-of-range, packet loss	Packet header, Adjacent node
4	3	circuit	Oversized packet loss	Packet header, Adjacent node
4	4	circuit	Packet format error	Packet beginning, Adjacent node
4	5	circuit	Partial routing update loss	Packet header, Highest address, Adjacent node
4	6	circuit	Verification reject	Node
4	7	circuit	Circuit down, circuit fault	Reason, Adjacent node
4	8	circuit	Circuit down	Reason,Packet header, Adjacent node
4	9	circuit	Circuit down, operator initiated	Reason, Packet header, Adjacent node
4	1	circuit	Circuit up	Adjacent node
4	1	circuit	Initialization failure, line fault	Reason
4	1	circuit	Initialization failure, operator fault, Reason, Packet header, Received version	Reason
4	1	node	Node reachability change	Status
4	1	circuit	Adjacency up	Adjacent node
4	1	circuit	Adjacency rejected	Adjacent node, Reason
4	1	area	Area reachability change	Status
4	1	circuit	Adjacency down	Reason, Packet header, Adjacent node
4	1	circuit	Adjacency down, operator initiated, Reason, Packet header	Adjacent node
5	0	circuit	Locally initiated state change	Old state, New state
5	1	circuit	Remotely initiated state change	Old state, New state
5	2	circuit	Protocol restart received in maintenance mode	None
5	3	circuit	Send error threshold	Circuit counters
5	5	circuit	Select error threshold	Circuit counters
5	6	circuit	Block header format error	Header (optional)
5	1	line	Send failed	Failure reason, Distance

[1]In this context, entity refers to a component or software module that can generate events.

(continued on next page)

Table 4–2 (Cont.): Event Log Summary

Class	Type	Entity[1]	Standard text	Counters
5	1	line	Receive failed	Failure reason, Ethernet header
5	1	line	Collision detect check failed	none

[1]In this context, entity refers to a component or software module that can generate events.

Chapter 5

Network Counters

The network software maintains counters for circuits, lines, and nodes. This chapter lists all counters in alphabetical order within component groups. In some cases, the counters respond to and reflect network events. In other cases, the counters respond to and reflect normal activities such as messages sent and messages received. Individual counter descriptions indicate whether that counter increments when a corresponding event occurs (see Chapter 4 for a description of event messages). Where possible, the description includes reasons why each of the counters might be incremented.

A counter does not display a value greater than its maximum value. When a counter overflows, it locks on the overflow value until it is zeroed. Counter displays with an angle bracket (>) indicate that the counter has overflowed. For example, if the maximum value for a counter is 255, its overflow display is >255. Each of the counter descriptions in this appendix includes the maximum value of the counter.

Each category of counters maintains a timing counter (seconds since last zeroed) that is zeroed when its associated counters are zeroed and starts when they start. In this way, the timing counter logs the seconds since its associated counters were zeroed to provide a time frame for them.

5.1 Circuit Counters

Circuit counters for DECnet–ULTRIX are maintained in the Network Management layer, the Routing layer, and the Data Link layer.

5.1.1 Network Management Layer

Seconds since last zeroed

This counter is zeroed when the other circuit counters are zeroed. It then increments by 1 every second so as to provide a time frame for the other circuit counters. The overflow value is 65,535.

5.1.2 Routing Layer

Circuit down

This counter records the number of times that a circuit was declared down by the executor. The overflow value is 255.

Initialization failure

This counter increments when the circuit could not be initialized by the executor for network use. The overflow value is 255.

Originating packets sent

This counter records the number of packets sent by the executor over the circuit. The overflow value is 4,294,967,295.

Terminating congestion loss

This counter records the number of packets intended for the node that were discarded because the Routing layer could not buffer them. The overflow value is 65,535.

Terminating packets received

This counter records the number of packets received by the executor with the executor as the destination. The overflow value is 4,294,967,295.

Transit congestion loss

This counter records the number of packets received by the executor that were to be routed to another node but were discarded because of heavy traffic on the output circuit. The overflow value is 65,535.

Transit packets received

This counter increments when the executor receives a packet that is to be routed to another node. The overflow value is 4,294,967,295.

Transit packets sent

This counter increments when the executor sends a packet through to another node. The overflow value is 4,294,967,295.

5.1.3 Data Link Layer

Bytes received

This counter increments when a data byte is received on the circuit. The count does not include Data Link Protocol overhead or bytes retransmitted by the Data Link layer. It can be used with the Data blocks received counter to determine the inbound traffic load on the circuit. The overflow value is 4,294,967,295.

Bytes sent

This counter increments when a data byte is sent on the circuit. The count does not include Data Link Protocol overhead or bytes retransmitted by the Data Link layer. It can be used with the **Data blocks sent** counter to determine the outbound traffic load on the circuit. The overflow value is 4,294,967,295.

Data blocks received

This counter increments when a data block is received on the circuit. The count does not include Data Link Protocol overhead. This counter can be used as a statistical base for evaluating the other Data Link layer counters. The overflow value is 4,294,967,295.

Data blocks sent

This counter increments when a data block is sent on the circuit. The count does not include Data Link Protocol overhead or blocks retransmitted by the Data Link layer. It can be used as a statistical base for evaluating the other Data Link layer counters. The overflow value is 4,294,967,295.

Data errors inbound

This counter indicates the number of incoming data errors on the circuit. It can have either of the following qualifiers: negative acknowledgments (NAKs) sent, data field block check error NAKs sent, reply response. The overflow is 255.

Data errors outbound

This counter indicates the number of outgoing data errors on the circuit. It can have any of the following qualifiers: NAKs received, header block check error NAKs received, data field block check error NAKs received, reply response. The overflow is 255.

Local buffer errors

This counter increments when a negative acknowledgment (NAK) is sent. It can have either of the following qualifiers: NAKs sent, buffer unavailable; NAKs sent, buffer too small. The overflow is 255.

Local reply timeouts

This counter increments each time a message is retransmitted because the retry timer for a sent message expired before a positive acknowledgment (ACK) was received from the remote node. The overflow value is 255.

Remote buffer error

This counter increments when a negative acknowledgment (NAK) is received. It can have either of the following qualifiers: NAKs received, buffer unavailable; NAKs received, buffer too small. The overflow value is 255.

Remote reply timeouts

This counter increments each time a message is retransmitted because the retry timer for a sent message expired before a positive acknowledgment (ACK) from your node was received at the remote node. The overflow value is 255.

Selection intervals elapsed

This counter records the number of times that the executor turned a circuit around or selected an adjacent node on both half-duplex and multipoint circuits. This counter is used as a statistical base for the evaluation of the counter for selection timeouts. The overflow value is 65,535.

Selection timeouts

This counter records the number of times that the executor turned a circuit around or selected an adjacent node on both half-duplex and multipoint circuits but the adjacent node failed to respond within the required time. This can be caused by blocks being lost on the circuit in either direction or by too small a value being specified for the executor's response timer. Blocks are usually lost because of a partial, temporary, or total failure of the communications line. This counter can have the following qualifier: No reply to select. The overflow value is 255.

User buffer unavailable

This counter indicates the total number of times that a user buffer was not available for an incoming frame that passed all filtering. User buffers are supplied by users on receive requests. The overflow value is 65,535.

5.2 Line Counters

Line counters for DECnet–ULTRIX are maintained in the Network Management layer and the Data Link layer.

5.2.1 Network Management Layer

Seconds since last zeroed

This counter is zeroed when the other line counters are zeroed. It then increments by 1 every second so as to provide a time frame for the other line counters. The overflow value is 65,535.

5.2.2 Data Link Layer

Blocks sent, initially deferred

This counter indicates the total number of times that a frame transmission was deferred on its first transmission attempt. It is used to measure Ethernet contention with no collisions. The overflow value is 4,294,967,295.

Blocks sent, multiple collisions

This counter indicates the total number of times that a frame was successfully transmitted on the third or later attempt after normal collisions had occurred on previous attempts. The overflow value is 4,294,967,295.

Blocks sent, single collision

This counter indicates the total number of times that a frame was successfully transmitted on the second attempt after a normal collision had occurred on the first attempt. The overflow value is 4,294,967,295.

Bytes received

This counter increments when a data byte is received on the line. The count does not include Data Link Protocol overhead or bytes retransmitted by the Data Link layer. It can be used with the **Data blocks received** counter to determine the inbound traffic load on the line. The overflow value is 4,294,967,295.

Bytes sent

This counter increments when a data byte is sent on the line. The count does not include Data Link Protocol overhead or bytes retransmitted by the Data Link layer. It can be used with the **Data blocks sent** counter to determine the outbound traffic load on the line. The overflow value is 4,294,967,295.

Collision detect check failure

This counter indicates the approximate number of times that a collision detect was not sensed after a transmission. The overflow value is 65,535.

Blocks received

This counter increments when a data block is received on the line. The count does not include Data Link Protocol overhead. It can be used as a statistical base for evaluating the other Data Link layer counters. The overflow value is 4,294,967,295.

Data blocks sent

This counter increments when a data block is sent on the line. The count does not include Data Link Protocol overhead or blocks retransmitted by the Data Link layer. It can be used as a statistical base for evaluating the other Data Link layer counters. The overflow value is 4,294,967,295.

Data overrun

This counter indicates the total number of times that the hardware lost an incoming frame because it was unable to keep up with the data rate. The overflow value is 65,535.

Local station errors

This counter records occurrences caused by a fault in a remote station or by an undetected error on the channel inbound to this station. The overflow value is 255. When this counter has a nonzero value, the type of failure is listed:

Local receive overrun (LOVRN)

The local station experienced a receive overrun and sent out a negative acknowledgment (NAK).

Local receive overrun (LOVR)

The local station experienced a receive overrun but did not send out a NAK.

Local transmit underruns (LUNDR)

The local station experienced a transmit underrun.

Local message header format error (LMHFE)

The local station sent a packet with a bad header for which the remote station sent out a NAK.

Multicast blocks received

This counter indicates the total number of multicast blocks that have been successfully received. The overflow value is 4,294,967,295.

Multicast bytes received

This counter indicates the total number of multicast data bytes that have been successfully received (including bytes in the Ethernet data field but not the Ethernet data link headers). The overflow value is 4,294,967,295.

Receive failure

This counter indicates the total number of blocks received with some data error. (The blocks are data frames that passed either physical or multicast address comparison.) The overflow value is 65,535. When this counter has a nonzero value, the type of failure that has occurred is also listed:

Block check error

The frame failed the cyclic redundancy check (CRC). The problem can be with either the local node or the remote node. The failure can be caused by electromagnetic interference, late collisions, or a faulty hardware controller. Use both circuit-level loopback tests to see whether your node is working correctly.

Framing error

The frame did not contain an integral number of 8-bit bytes. The problem can be with either the local node or the remote node. The failure can be caused by electromagnetic interference, late collisions, or a faulty hardware controller. Use both circuit-level loopback tests to see whether your node is working correctly.

Frame too long

The frame was discarded because it was either longer than the maximum or shorter than the minimum allowable length for the Ethernet. The remote node is sending frame lengths that do not meet the requirements of the Ethernet specification. The problem could be with the DEUNA/DEQNA, the H4000 transceiver, the transceiver cable, the DELNI, or the transmitting node.

Remote station error

This counter records occurrences caused by a fault in a remote station or by an undetected data error on the channel inbound to this station. The overflow value is 255. When this counter has a nonzero value, the type of failure that has occurred is listed:

Remote receive overrun (ROVRN)

The remote station experienced a receive overrun and sent out a negative acknowledgment (NAK).

Remote message header format errors (RMHFE)

The remote station sent a packet with a bad header for which the local station sent out a NAK.

Remote streaming tributaries (RSTR)

The remote station failed to release the channel at the end of the selection interval, or the maximum transmission interval (different for each implementation) is exceeded without releasing the channel.

Send failure

This counter usually indicates the total number of times that a transmit attempt failed. However, this counter can also increment on successful transmissions when a DECOM (broadband transceiver) and DEQNA controller are used together. The overflow value is 65,535. When this counter indicates a failure, the type of failure that has occurred is also listed:

Carrier check failed

The data link did not sense a signal that must accompany transmission of a frame. This condition indicates a failure during transmission because of a problem with the DEUNA/DEQNA, the H4000 transceiver, or the transceiver cable.

Excessive collisions

The maximum number of retransmissions resulting from collisions during transmissions has been exceeded. Transmissions are failing because frames being sent are colliding with frames being transmitted by other nodes. This condition can occur if the network is overloaded or if there is a hardware problem.

Frame too long

Either the DEUNA/DEQNA controller or the transceiver truncated the frame at the maximum buffer size. Either your node tried to send a frame that was too long, or the transceiver cut off the message too soon.

Open circuit

There is a break somewhere along the communications path. If this problem exists on your node only, it is probably a fault with the DEUNA/DEQNA option module, the H4000 transceiver, or the DELNI. In this case, use the loopback tests to isolate the problem. If other nodes report the same problem, the fault is probably with an H4000 transceiver connection, a DELNI connection, or the Ethernet cable itself.

Remote failure to defer

A remote node began transmitting while your node was still actively transmitting. Either there is a problem with the remote node's carrier sense, or there is a weak transmitter on your node. Use the loopback test with transmit assistance to determine whether the remote node can detect a transmission from the assistant node.

Short circuit

There is a short circuit in either the Ethernet cable, the H4000 transceiver, the DELNI, or the DEUNA/DEQNA option module. If this problem exists on your node only, it is probably a faulty DEUNA/DEQNA. In this case, use the loopback tests to isolate the problem.

System buffer unavailable

This counter indicates the total number of times that no system buffer was available for an incoming frame. This can apply to any buffer between the hardware and the user buffers (those supplied on receive requests). The overflow value is 65,535.

Unrecognized frame destination

This counter indicates the number of times that a frame was discarded because there was no enabled portal with the protocol type or multicast address. The count includes frames received for the physical address, broadcast address, or multicast address. The overflow value is 65,535.

User buffer unavailable

This counter indicates the number of times no user buffer was available for an incoming frame that passed all filtering. The user buffer is one supplied by the user on a receive request.

5.3 Node Counters

Node counters for DECnet–ULTRIX are maintained in the Network Management layer and the End Communications layer. Additional counters are kept for the executor node.

5.3.1 Network Management Layer

Seconds since last zeroed

This counter is zeroed when the other node counters are zeroed. It then increments by 1 every second so as to provide a time frame for the other node counters. The overflow value is 65,535.

5.3.2 End Communications Layer

Buffer unavailable

This counter indicates the total number of data segments discarded due to insufficient cache buffering. The overflow value is 65,535.

Connects received

This counter increments when a connect initiation signal is received from the associated node. The overflow value is 65,535.

Connects sent

This counter increments when a connect initiation signal is sent to the associated node. The overflow value is 65,535.

Response timeouts

This counter increments when the associated node fails to respond within the required time. This situation can be caused either by messages being discarded in the network or by a wide variance in the round-trip delay to the node. This condition normally indicates an overload condition in the network. This should be considered a problem if 2 percent or more of the messages sent are timed out. The overflow value is 65,535.

Total bytes received

This counter increments when bytes are received from the associated node at the logical link level. The count includes bytes from both user messages and logical link protocol messages. The overflow value is 4,294,967,295.

Total bytes sent

This counter increments when bytes are sent to the associated node at the logical link level. The count includes bytes from both user messages and logical link protocol control messages. The overflow value is 4,294,967,295.

Total messages received

This counter increments when a message is received from the associated node at the logical link level. The count includes both user messages and logical link protocol control messages. Furthermore, it includes internal segmentation of user messages by the Network Services layer. The overflow value is 4,294,967,295.

Total messages sent

This counter increments when a message is sent to the associated node at the logical link level. The count includes both user messages and logical link protocol control messages. It also includes the retransmission of a message. The Network Services layer segments the user messages. The overflow value is 4,294,967,295.

User bytes received

This counter increments when user data bytes are received from the associated node at the logical link level. It includes only the user data from data messages and from interrupt, connect, accept, reject, disconnect, and abort functions. The overflow value is 4,294,967,295.

User bytes sent

This counter increments when user data bytes are sent to the associated node at the logical link level. It includes only the acknowledged user data from data messages and from interrupt, connect, accept, reject, disconnect, and abort functions. It does not include retransmissions. The overflow value is 4,294,967,295.

User messages received

This counter increments when a user message is received from the associated node at the logical link level. The overflow value is 4,294,967,295.

User messages sent

This counter increments when a user message is sent to the associated node at the logical link level. The overflow value is 4,294,967,295.

5.3.3 Executor Node Counters

Aged packet loss

This counter increments when a packet is discarded because it has visited too many nodes. The count is the total of all such discards by the executor node. This counter is incremented each time the aged packet loss event occurs. The overflow value is 255.

Node out-of-range packet loss

This counter increments when a packet is discarded because the destination node address was greater than the maximum address defined for the executor. The count is the total of all such discards by the executor node. This counter is incremented each time the node out-of-range packet loss event occurs. The overflow value is 255.

Node unreachable packet loss

This counter increments when a packet is discarded because its destination node was unreachable. The count is the total of all such discards by the executor node. The counter is incremented each time the node unreachable packet loss event occurs. The overflow value is 65,535.

Oversized packet loss

This counter increments when a packet is discarded because it was larger than the circuit buffer size. The circuit buffer size was previously established between the executor node and the adjacent node. The counter is incremented each time the oversized packet loss event occurs. The overflow value is 255.

Packet format error

This counter increments when a packet is discarded because of invalid packet control information. The count is the total of all such discards by the executor node. This counter is incremented each time the packet format error event occurs. The overflow value is 255.

Partial routing update loss

This counter increments when part of a routing update is lost because it contained a reachable node address that exceeded the maximum address defined for the executor node. The count is the total of all such occurrences at the executor node. Only routing nodes keep this counter. This counter is incremented each time the partial routing update loss event occurs. The overflow value is 255.

Peak logical links active

This counter records the number of active logical links between the executor and all nodes (including itself). The overflow value is 65,535.

Verification reject

This counter increments when the executor rejects a verification request from an adjacent node during routing initialization. The count is the total of all such occurrences at the executor node. This counter is incremented each time the verification reject event occurs. The overflow value is 255.

Appendix A

Command Summary

This appendix summarizes the **ncp** commands supported by DECnet–ULTRIX.

You may wish to review the graphic conventions listed in the Preface, especially the use of braces { }, brackets [ ], and parentheses ( ). This summary also uses the following notations:

* = Command cannot be used with the **tell** prefix.

S = Command can be used by someone with superuser privileges only.

A = Command can be used by all users.

S **clear circuit** *circuit-id* [**receive password** | **transmit password**]

S **clear executor** { **identification** | **incoming timer** | **outgoing timer** }

A* **clear executor node**

S **clear** { **known logging** | **logging console** | **logging file** | **logging monitor** } [**name** | **events** *event-list* | **known events** [**circuit** *circuit-id* | **line** *line-id* | **node** *node-id*]] [**sink** { **executor** | **node** *node-id* }]

S clear { node *node-id* / known nodes } { diagnostic file / dump file / hardware address / host / load file / name / secondary loader / service circuit / service password / tertiary loader }

or

clear { node *node-id* / known nodes } all

S clear { object *object-name* / known objects }

S define circuit *circuit-id* { hello timer *seconds* / receive password *password* / service { disable / enable } / state *circuit-state* { off / on } / transmit password *password* }

S define executor { address *node-address* / delay factor *number* / delay weight *number* / gateway access { disable / enable } / gateway user *login-name* / identification *id-string* / inactivity timer *seconds* / incoming proxy { disable / enable } / incoming timer *seconds* / maximum links *number* / maximum node counters *number* / name *node-name* / outgoing proxy { disable / enable } / outgoing timer *seconds* / pipeline quota *number* / retransmit factor *number* / segment buffer size *number* }

S define line *line-id* { controller { loopback | normal } | duplex { full | half } | protocol { ddcmp dmc | ddcmp point | ethernet } }

S define { known logging | logging console | logging file | logging monitor } [name *name* | state { off | on } | events *list* | known events | [circuit *circuit-id* | line *line-id* | node *node-id*] | [sink { executor | node *node-id* }]]

S define node *node-id* { address *node-address* | diagnostic file *file* | dump file *file* | hardware address *E-address* | host *node-id* | load file *file* | name *node-name* | secondary [loader] *file* | service { disable | enable } | service circuit *circuit-id* | [service] password *service-password* | tertiary [loader] *file* }

S define { object *object-name* | known objects } { accept { deferred | immediate } | default user *login-name* | file *file-id* | number *number* | type { sequenced packet | stream } }

A list { circuit *circuit-id* | known circuits } [characteristics | status | summary]

A list executor [characteristics | status | summary]

A **list** { **line** *line-id* / **known lines** } [**characteristics** / **status** / **summary**]

A **list** { **known logging** / **logging console** / **logging file** / **logging monitor** } [**characteristics** / **status** / **summary** / **events**] [**known sinks** / **sink node** *node-id*]

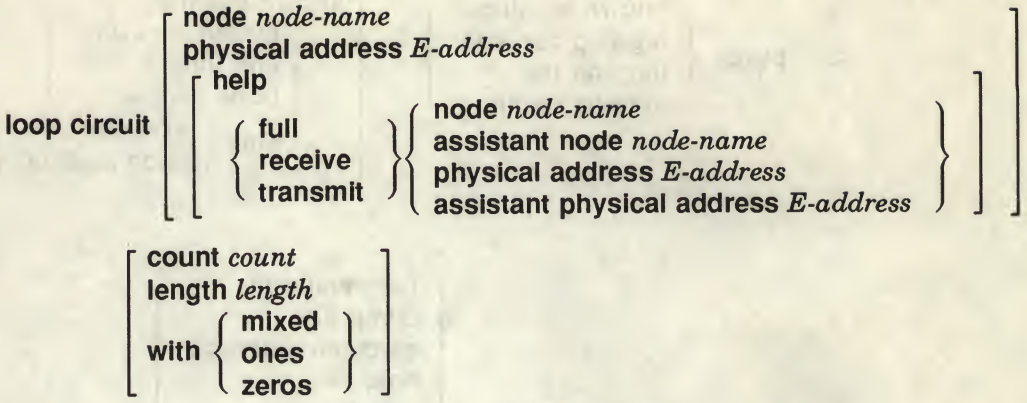
A **list** { **node** *node-id* / **known nodes** } [**characteristics** / **status** / **summary**]

A **list** { **object** *object-name* / **known objects** } [**characteristics** / **summary**]

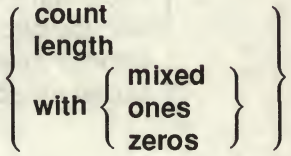
S **load node** *node-id* { **address** *node-address* / **from** *load-file* / **host** *node-id* / **name** *node-name* / **physical address** *E-address* / **secondary [loader]** *file* / **[service] password** *service-password* / **tertiary [loader]** *file* / **via** *circuit-id* }

S **load via** *circuit-id* { **address** *address* / **from** *load-file* / **host** *node-id* / **name** *node-name* / **physical address** *E-address* / **secondary [loader]** *file* / **[service] password** *service-password* / **tertiary [loader]** *file* }

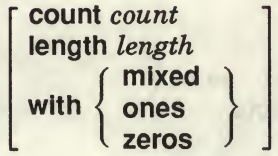
S



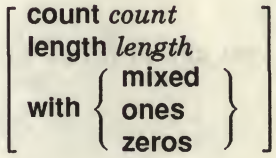
S loop line *line-id*



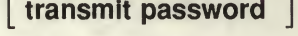
A loop executor



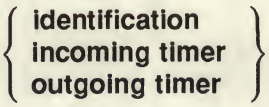
A loop { node *node-id*[*acc-con-info*] }

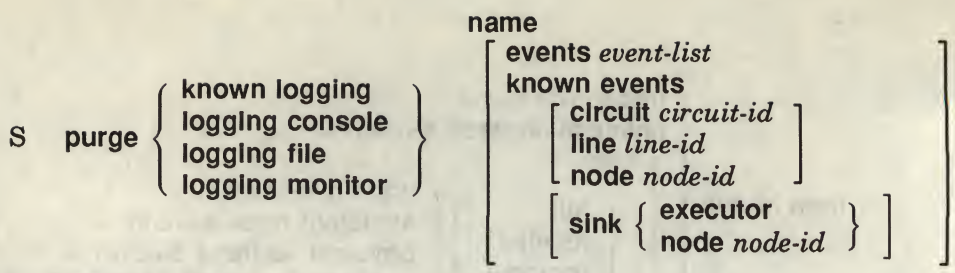


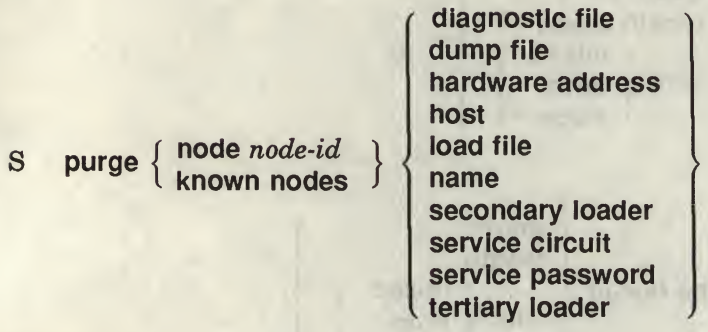
S purge circuit *circuit-id*



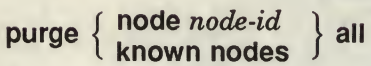
S purge executor

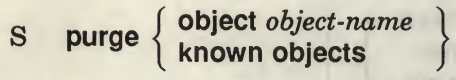


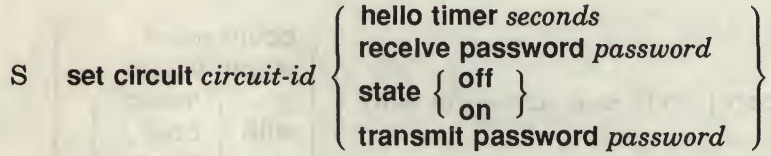

S
purge
known logging
logging console
logging file
logging monitor
name
events event-list
known events
circuit circuit-id
line line-id
node node-id
sink
executor
node node-id


S
purge
node node-id
known nodes
diagnostic file
dump file
hardware address
host
load file
name
secondary loader
service circuit
service password
tertiary loader

or


purge
node node-id
known nodes
all


S
purge
object object-name
known objects


S
set circuit circuit-id
hello timer seconds
receive password password
state
off
on
transmit password password

or


set circuit circuit-id all

S set executor
- address *node-address*
- delay factor *number*
- delay weight *number*
- gateway access { disable | enable }
- gateway user *login-name*
- identification *id-string*
- inactivity timer *seconds*
- incoming proxy { disable | enable }
- incoming timer *seconds*
- maximum links *number*
- maximum node counters *number*
- name *node-name*
- outgoing proxy { disable | enable }
- outgoing timer *seconds*
- pipeline quota *number*
- retransmit factor *number*
- segment buffer size *number*
- state { on | off | shut | restricted }

or

set executor all

A* set executor node *node-id*[*acc-con-info*]

S set line *line-id*
- controller { loopback | normal }
- duplex { full | half }
- protocol { ddcmp dmc | ddcmp point | ethernet }

or

set line *line-id* all

S set { known logging | logging console | logging file | logging monitor }
- name *name*
- state { off | on }
- events *list*
- known events
  - [circuit *circuit-id* | line *line-id* | node *node-id*]
  - [sink { executor | node *node-id* }]

S set { node *node-id* } { address *node-address* | diagnostic file *file* | dump file *file* | hardware address *E-address* | host *node-id* | load file *file* | name *node-name* | secondary [loader] *file* | service circuit *circuit-id* | [service] password *service-password* | tertiary [loader] *file* }

or

set { node *node-id* } all

S set { object *object-name* | known objects } [accept { deferred | immediate } | default user *login-name* | file *file-id* | number *number* | type { sequenced packet | stream }]

or

set { object *object-name* | known objects } all

A show { circuit *circuit-id* | known circuits } [characteristics | counters | status | summary]

A show executor [characteristics | counters | status | summary]

A show { line *line-id* | known lines } [characteristics | counters | status | summary]

A show { known logging | logging console | logging file | logging monitor } [characteristics | status | summary | events] [known sinks | sink node *node-id*]

A **show** { **node** *node-id* / **active nodes** / **known nodes** } [**characteristics** / **counters** / **status** / **summary**]

A **show** { **object** *object-name* / **known objects** } [**characteristics** / **summary**]

A* **tell** *node-id* [*acc-con-info*] *ncp-command*

S **trigger node** *node-id* { **physical address** *E-address* / **[service] password** *hex-password* / **via** *circuit-id* }

S **trigger via** *circuit-id* { **physical address** *E-address* / **[service] password** *service-password* }

S **zero circuit** *circuit-id* [counters]

S **zero executor [counters]**

S **zero line** *line-id* **[counters]**

S **zero** { **node** *node-id* / **known nodes** } **[counters]**

Appendix B

DECnet-Supplied Objects

This appendix lists and defines the DECnet–ULTRIX objects. Table B-1 lists these objects and gives the following information about them: object number, file name, default user, socket type, and accept mode.

Table B–1: Digital-Supplied DECnet–ULTRIX Objects

Object = tell	
Number	= 0
File	= /usr/bin/tell
Default user	=
Type	= Sequenced packet
Accept	= Immediate
Object = DEFAULT	
Number	= 0
File	=
Default user	=
Type	= Sequenced packet
Accept	= Immediate
Object = fal	
Number	= 17
File	= /usr/etc/fal
Default user	= guest
Type	= Sequenced packet
Accept	= Deferred
Object = nml	
Number	= 19
File	= /usr/etc/nml
Default user	= guest
Type	= Sequenced packet
Accept	= Deferred
Object = dterm	
Number	= 23
File	= /usr/etc/dtermd

(continued on next page)

Table B–1 (Cont.): Digital-Supplied DECnet–ULTRIX Objects

Default user	= guest
Type	= Stream
Accept	= Immediate
Object = mir	
Number	= 25
File	= /usr/etc/mir
Default user	= guest
Type	= Sequenced packet
Accept	= Deferred
Object = mail11	
Number	= 27
File	= /usr/etc/mail11dv3
Default user	= daemon
Type	= Sequenced packet
Accept	= Deferred
Object = dlogin	
Number	= 42
File	= /usr/etc/dlogind
Default user	= guest
Type	= Sequenced packet
Accept	= Immediate
Object = dtr	
Number	= 63
File	= /usr/etc/dtr
Default user	= guest
Type	= Sequenced packet
Accept	= Deferred

Table B-2 describes the services of Digital-supplied DECnet-ULTRIX objects.

Table B–2: DECnet–ULTRIX Object Descriptions

DECnet Object	Service
tell[1]	Utility that lets you execute commands on a remote DECnet–ULTRIX or DECnet–VAX node.
DEFAULT	You can use the DECnet zero object named to create your own network server programs without modifying the object database. If the DECnet object spawner does not find a match for an object in the object database, it searches for the program by using the path name specified in the **connect** request. The path name can be absolute (the full path name) or relative (the path defined by the user name and password).

[1]Not to be confused with the **tell** prefix used with **ncp** commands.

(continued on next page)

Table B–2 (Cont.): DECnet–ULTRIX Object Descriptions

DECnet Object	Service
	Object numbers are equivalent to object names and can be used on other networks unless the object number is zero. If the object number is zero, you must use an object name.
fal	Allows a process running on any node in a DECnet network to access another node's file system. The **fal** object uses the Data Access Protocol (DAP) to communicate with other nodes.
nml	The process that performs network management services. The **nml** object communicates with other nodes in the network by means of the NICE protocol.
dterm	A homogeneous command terminal service. The **dterm** object uses the TOPS-20 protocol.
mir	The remote loopback mirror, which performs looping services for remote users.
mail11	The DECnet mail service.
dlogin	The heterogeneous command terminal service. The **dlogin** object uses the CTERM protocol, which supports connections from DECnet–ULTRIX to all other DECnet nodes.
dtr	The DECnet Test Receiver that is used with the DECnet Test Sender (**dts**) to test logical links.

Appendix C

Ethernet Addressing

A unique Ethernet address identifies each node on an Ethernet line. You can send a message to any number of nodes on an Ethernet line, depending on the type of Ethernet address you use, physical or multicast.

To configure your network you need not specify the Ethernet address of a node. Whenever you execute the **ncp set executor state on** command, DECnet resets the Ethernet physical address to correspond to the DECnet node address.

Whenever the DECnet software changes the Ethernet address of your Ethernet controller, such as during DECnet startup, it invalidates the existing Internet Ethernet address mapping for the node. The ULTRIX Ethernet driver detects the change in address in 5 minutes. If you want to delete the mapping entry in the address translation table manually, use the **arp -d** command. See the ULTRIX **arp(8)** manual page.

C.1 Ethernet Address Format

Ethernet addresses are represented as six pairs of hexadecimal digits separated by hyphens (for example, AA-00-03-00-67-FF).

Xerox Corporation assigns a block of addresses to a producer of Ethernet interfaces upon application. Thus, every manufacturer has a unique set of addresses to use. Normally, one address out of the assigned block of physical addresses is permanently associated with each interface (usually in a read-only memory). This address is known as the *Ethernet hardware address* of the interface.

NOTE

You can use the **show line** *line-id* **characteristics** command to display the hardware address.

Digital's interface to Ethernet (the DEUNA or DEQNA controller at the node) has the ability to set a different logical address to be used by the interface. This address is known as the *Ethernet physical address*. When a node on the Ethernet initially starts up, the physical address is the same as the Ethernet hardware address. Then, when DECnet turns on a DEUNA or DEQNA device, DECnet constructs a physical address by appending the local node's node address to a constant 8-digit number derived from the block addresses assigned to Digital (AA-00-04-00).

Once the Ethernet physical address has been set to its new value, it is reset to its original hardware address value only when a reset is issued to the DEUNA or DEQNA (for example, when the machine power is shut off).

C.2 Ethernet Multicast Address Types

Ethernet physical addresses and Ethernet multicast addresses are distinguished by the value of the leading low-order bit of the first byte of the address:

- **Physical address.** The unique address of a single node on any Ethernet (low-order bit = 0).
- **Multicast address.** A multidestination address of one or more nodes on a given Ethernet (low-order bit = 1).

There are two types of multicast addresses:

- A **multicast group address** is an address that is assigned to any number of node groups so that they are all able to receive the same message in a single transmission by a sending node.
- A **broadcast address** is a single multicast group address (specifically, FF-FF-FF-FF-FF-FF) to which a message can be sent if it must be received by all nodes on a given Ethernet. (Use a broadcast address only for messages to be acted upon by all nodes on the Ethernet, since all nodes must process them.)

C.3 Ethernet Physical and Multicast Address Values

Digital physical addresses are in the range AA-00-00-00-00-00 through AA-00-04-FF-FF-FF. Multicast group addresses assigned for use in cross-company communications are as follows:

Value	Meaning
FF-FF-FF-FF-FF-FF	Broadcast
CF-00-00-00-00-00	Loopback assistance

Digital multicast group addresses assigned to be received by other Digital nodes on the same Ethernet are as follows:

Value	Meaning
AB-00-00-01-00-00	Dump/load assistance
AB-00-00-02-00-00	Remote console
AB-00-00-03-00-00	All Phase IV routers
AB-00-00-04-00-00	All Phase IV end nodes
AB-00-00-05-00-00	Reserved for future use
through	
AB-00-04-FF-FF-FF	

Index

M

N

O

P

R

S

T

Z

HOW TO ORDER ADDITIONAL DOCUMENTATION

DIRECT TELEPHONE ORDERS

In Continental USA
call 800–DIGITAL

In Canada
call 800–267–6215

In New Hampshire
Alaska or Hawaii
call 603–884–6660

In Puerto Rico
call 809–754–7575

ELECTRONIC ORDERS (U.S. ONLY)

Dial 800–DEC–DEMO with any VT100 or VT200
compatible terminal and a 1200 baud modem.
If you need assistance, call 1–800–DIGITAL.

DIRECT MAIL ORDERS (U.S. and Puerto Rico*)

DIGITAL EQUIPMENT CORPORATION
P.O. Box CS2008
Nashua, New Hampshire 03061

DIRECT MAIL ORDERS (Canada)

DIGITAL EQUIPMENT OF CANADA LTD.
940 Belfast Road
Ottawa, Ontario, Canada K1G 4C2
Attn: A&SG Business Manager

INTERNATIONAL

DIGITAL
EQUIPMENT CORPORATION
A&SG Business Manager
c/o Digital's local subsidiary
or approved distributor

Internal orders should be placed through the Software Distribution Center (SDC),
Digital Equipment Corporation, Westminster, Massachusetts 01473

*Any prepaid order from Puerto Rico must be placed
with the Local Digital Subsidiary:
809–754–7575 x2012

HOW TO ORDER ADDITIONAL DOCUMENTATION

DIRECT TELEPHONE ORDERS

ELECTRONIC ORDERS (U.S. ONLY)

DIRECT MAIL ORDERS (U.S. and Puerto Rico)

DIRECT MAIL ORDERS (Canada)

INTERNATIONAL

DECnet–ULTRIX
NCP Command Reference
AA–PB62A–TE

READER'S COMMENTS

What do you think of this manual? Your comments and suggestions will help us to improve the quality and usefulness of our publications.

Please rate this manual:

	Poor				Excellent
Accuracy	1	2	3	4	5
Readability	1	2	3	4	5
Examples	1	2	3	4	5
Organization	1	2	3	4	5
Completeness	1	2	3	4	5

Did you find errors in this manual? If so, please specify the error(s) and page number(s).

_____

_____

_____

_____

General comments:

_____

_____

_____

_____

Suggestions for improvement:

_____

_____

_____

_____

Name _____ Date _____

Title _____ Department _____

Company _____ Street _____

City _____ State/Country _____ Zip _____

DO NOT CUT – FOLD HERE AND TAPE

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY LABEL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

digital™

Networks and
Communications Publications
550 King Street
Littleton, MA 01460–1289

DO NOT CUT – FOLD HERE

CUT ON THIS LINE

digital